Blockchain sentiment on sector analysis: LSTM approach

Master's Thesis submitted

to

Prof. Cathy Yi-Hsuan Chen

Humboldt-Universität zu Berlin
School of Business and Economics
Institute for Statistics and Econometrics
Chair of Econometrics

Dr. Jens Kolbe

Technische Universität Berlin
Faculty VII - Economics and Business
Chair of Statistics

by

Alexander Döbele
(566824)

in partial fulfillment of the requirements
for the degree of
Master of Science
Berlin, July 15, 2019

Abstract

In this thesis, an LSTM architecture is presented to perform sentence-level sentiment analysis for blockchain news articles. An in-depth look into the model specification and the choice of different model parameters is given. Around 25000 articles from April 2013 to April 2019 were gathered from the business section of Coindesk (2019) using a dynamic Web-scraper. Various methods like SMOTE oversampling are introduced to account for an imbalanced training dataset. The model performance is compared to an SVM model, and it could be shown that LSTM is better suited for sentiment analysis as it is superior in modelling long term semantic dependencies in text data.

Furthermore, a static contemporaneous Regression is employed to explain the relationship between blockchain sentiment and returns in different US Sectors. After identifying the most promising US sectors with stronger ties to the blockchain news sentiment, the regression model could not establish a significant influence of blockchain sentiment on US sector returns. However, the hypothesis that blockchain sentiment today increases the conditional volatility in returns tomorrow could be established for specific sectors by implementing a GARCH model.

Zusammenfassung

Die folgende Arbeit beschäftigt sich mit einer Machine learning Applikation aus dem Bereich der neuronalen Netze. Ein LSTM Model wird genutzt, um eine Sentiment-Analyse im Bereich blockchain Nachrichten durchzuführen. Dabei wird dem Leser ein tieferer Einblick in die Model-Spezifikationen und Wahl der Parameter gegeben. Es wurden insgesamt ca. 25000 Artikel im Zeitraum von April 2013 bis April 2019 von Coindesk (2019) mit Hilfe eines dynamischen Web-scrapers gesammelt. Außerdem werden verschiedene Methoden vorgestellt, die strukturelle Probleme im trainings Datensatz, wie beispielsweise "imbalanced Data", beheben sollen. Zu diesem Zweck wird das Konzept des "SMOTE oversamplings" vorgestellt. Anschließend werden die Ergebnisse des LSTM Models mit den Ergebnissen eines anderen Models aus dem Bereich Machine learning (SVM) verglichen. Es konnte gezeigt werden, dass sich LSTM für Sentiment-Analysen besser eignet, da es in der Lage ist Abhängigkeiten über einen längeren Zeitraum in Textdokumenten besser abzubilden. Weiterhin wird eine statische Regression vorgestellt, die den Zusammenhang zwischen den geschätzten Sentiment Indizes und dem Ertrag verschiedener US Sektoren erklären soll. Nachdem die vielversprechensten Sektoren, mit stärkeren Einflüssen aus dem Bereich Blockchain, identifiziert wurden, konnte keine signifikante Beziehung zwischen Erträgen aus US Sektoren und Blockchain Sentiment etabliert werde. Jedoch konnte die Hypothese, dass Blockchain Sentiment heute die bedingte Varianz in Sektorerträgen morgen beeinflusst, mit Hilfe eines GARCH Models bestätigt werden.

Contents

List of Abbreviations

NLP	Natural language processing	DNN	Deep neural network
RNN	Recurrent neural network	LSTM	Long short term memory
BTC	Bitcoin	SVM	Support vector machine
ReLU	Rectified Linear Units	CNN	Convolutional neural network
FFNN	Feedforward neural network.	GRU	Gated recurrent unit
CBOW	Continuous bag of words	OMX	Office Max
HTML	Hyper Text Markup Language	BLEU	Bilingual Evaluation Understudy
SGD	stochastic gradient descent	SMOTE	Synthetic Minority Oversampling TEchnique

List of Figures

List of Tables

1 Introduction

1.1 Overview

Over the last years, the extent of unstructured text data overall and on the internet has flourished rapidly. Therefore the demand to process and extract helpful information is increasing accordingly. Every sentence plays an essential part in determining the opinion of an individuum towards a particular topic, composing the semantics of single words to form an expression of a whole meaning (Yin, 2017). The interest for different technologies in natural language processing (NLP) like sentiment analysis in social media or gathering knowledge from big unstructured data on the internet has been increasing drastically. As society relies more and more on the merits of the world wide web, the potential of gathering information by NLP is rising accordingly. The vast majority of NLP problems were lead by shallow machine learning methods focusing on in-depth feature engineering. The renaissance of deep neural networks (DNN's) facilitates the possibility to deal with NLP problems via deep systems with no or less artificial features (Yin, 2017). A neural network architecture that has elevated the space is the recurrent neural network (RNN). RNNs are a special kind of neural network that takes sequential input and produces following output by sharing parameters between time steps (Mikolov et al., 2010). The benefits of RNNs have shown to be most useful in natural language processing (Socher et al., 2011), image captioning (Mao et al., 2014), and speech recognition (Graves et al., 2013).

1.2 Contribution

This thesis focuses on one specific field of NLP, which is sentence based sentiment analysis or opinion mining in the context of blockchain news. According to Aboody et al. (2018) sentiment plays a crucial role in determining price evolution, given a possible arbitrage opportunity and intangible fundamental values. Therefore 25000 articles from Coindesk (2019) over the last six years have been gathered by a dynamic Web-Scraper. The blockchain technology is an open ledger for coordination, record keeping and irrevocability of transactions (Swan, 2015). It gained traction in the past couple of years due to a spike in the price of its most popular application Bitcoin. To carry out this task, a specific form of RNN's, the longterm short-term memory (LSTM), is used. LSTMs have been proven to be very useful to model word sequences without assuming word order independence and are powerful to

learn on data with long-range temporal dependencies (Zhou et al., 2016). Different performance measures of the LSTM and other machine learning approaches (SVM) are analyzed and compared. The resulting estimated sentiment indices are used to determine the effect of a change in sentiment towards blockchain technology on the returns and the conditional variance of different U.S. sectors and industries. As Bitcoin (Nakamoto et al., 2008) only recently introduced blockchain to a broader audience, one might expect a more significant influence on US sectors that already implied the technology or are planning to do so.

1.3 Thesis contents

The remaining of this document is organized as follows. Chapter 2 introduces basic concepts from the field of sentiment analysis. Different types, concerning the level of analysis, are presented. Followed by a short overlook of approaches to sentiment analysis, like Lexical analysis and Machine learning analysis.

Chapter 3 introduces the reader to deep learning methods and algorithms. The basic structure, as well as advantages and disadvantages, of commonly used neural networks, are shown while focusing on recurrent neural networks and LSTM in particular. The architecture of an LSTM is outlined, and the importance of word embeddings is emphasised. Afterwards, the LSTM algorithm setup is shown in detail, and the reasoning behind the choice of particular parameters is given. Necessary steps during the training process are highlighted, and the training dataset is introduced. Problems that came up during model training are identified, and possible solutions to those are given. The chapter concludes with a confusion matrix that shows performance results for the LSTM models and compares those to the results of an SVM model.

In Chapter 4, the dataset used for sentiment prediction is outlined, and the method to gather the data from Coindesk (2019) by a dynamic Web-Scraper is presented. Further, the necessary preprocessing of the data is described, and the method to construct a sentiment index based on the given data is shown. Chapter 5 introduces possible applications of the constructed sentiment index. First, the US Sector Data is preprocessed and control variables, to isolate the effect of sentiment on sector returns, are introduced. Afterwards, a static contemporaneous regression model is used to estimate the impact of blockchain sentiment on US sector returns. Finally, the influence of sentiment on the conditional volatility of returns in US sectors is analysed by using a GARCH model. Chapter 6 concludes this document by summarizing its main points and presenting directions for future work.

2 Sentiment Analysis

This chapter provides an overview of related concepts in the field of sentiment analysis. The first part focuses on different types of sentiment analysis and illustrates some critical steps in the process. In the second part, two classical approaches to sentiment analysis are introduced, and various metrics are given to evaluate their performance.

Sentiment analysis is an area of natural language processing (**NLP**) to determine the opinions and attitudes of an author towards a particular topic. With the rising popularity of blogs and social networks, opinion mining and sentiment analysis became a field of interest for many types of research. A comprehensive overview of the current work was presented in Pang et al. (2008). Sentiment describes an opinion or attitude declared by an individual, the opinion holder, about an entity, the target (Scheible, 2014).
According to Liu (2015), an opinion has two key components a target and a sentiment on the target. A target can be any entity or aspect of the entity on which an opinion has been expressed. The degree and direction of sentiment (i.e. positive, neutral and negative) are known as sentiment polarity. The most common polarity design focuses on only two categories, positive and negative, which establish the maximum and minimum on a continuous scale. A sentiment polarity defined that way represents most of the rating mechanisms found online:

- Thumbs up or down (Youtube and Facebook)
- Rating by stars (Tripadvisor, Amazon and IMDb)
- Positive, neutral and negative (eBay)

However, not every article or comment on a specific topic expresses an entirely positive or negative sentiment. Koppel and Schler (2006) state the importance of neutral sentiment and the additional information gain by accounting for it. Polarity is often used as the interval [-1,1] with -1 as perfect negative sentiment and 1 perfect positive sentiment. According to this scale, 0 describes perfect neutral sentiment. Although neutral sentiment can carry information in itself, some literature defines the task as a two-category problem (Dave et al., 2003).

2.1 Types of sentiment analysis

Sentiment classification is arguably the most widely studied topic in the field of sentiment analysis((Blitzer et al., 2007),(Aue and Gamon, 2005) and (Chesley et al., 2006)). The increasing interest leads to different ways to conduct sentiment analysis. The most noticeable difference lays in the level of granularity (Tripathy et al., 2017).

- Document-level analysis: As to whether the whole document expresses a positive or negative sentiment.
- Sentence level analysis: Determines whether the sentence expresses any negative, positive or neutral opinion.
- An Aspect-based analysis: Is mainly concerned with a particular aspect of the topic or product.

Document-level sentiment analysis assumes that the opinionated document expresses opinions on a single target, and the opinions belong to a single person (Sadegh et al., 2012). This assumption holds for customer reviews of products which commonly focus on one product and are written by only one reviewer (e.g. movies and restaurants). Its main task is to predict whether a reviewer wrote a positive or negative review, based on the text of the review. This standard text classification problem was addressed by Pang et al. (2002) using machine learning techniques, such as maximum entropy classification, Naive Bayes classifier and Support Vector Machines (SVM). Formally, document level classification is defined as follows (Liu et al., 2010). A Set of documents D is given. For each document d $\in$ D, the polarity of sentiment towards an object is determined. Given a document d which relates to an object o, determine the orientation oo of the opinion expressed on o, i.e., discover the opinion orientation oo on feature f in the quintuple (o, f, so, h, t), where f = o and h, t, o are assumed to be known or irrelevant. Assuming a document d (e.g. a product review) expresses opinions on a single object o and the opinions are from a single opinion holder h. This assumption holds for product reviews but might not hold for forum and blog posts as the author can express views on multiple products (Liu et al., 2010). Besides supervised machine learning techniques for document-level sentiment classification unsupervised methods are used in the literature as well (Turney, 2002).

The first step in *sentence level classification* is classifying a sentence as objective or subjective. In the literature, this task is called subjectivity classification (Sadegh et al., 2012).

Afterwards, the sentiment polarity of individual sentences is calculated. As both tasks are classification problems, traditional supervised training models are again applicable (Liu et al., 2010). The main challenge in applying those models is the manual effort of labelling a large number of training examples. The technique of automatically marking the training data via a bootstrapping approach is introduced by Riloff and Wiebe (2003). Note that the quintuple (o, f, oo, h, t) cannot be used to determine the task as sentence-level classification is an intermediate step. Determine which target or aspect has been specified in the sentence is the main subject in this classification. Without the target of a sentence, the classification for the sentence is useless.

One fundamental assumption of sentence-level classification focuses on the number of opinions and opinion holders in a sentence. There can only be a single opinion from a single opinion holder (Liu et al., 2010). This assumption only holds for simple sentences with a single opinion, e.g. "The panel of this television is excellent". A more complex sentence might violate this assumption, e.g. "The panel of this television is excellent, but the remote is hard to use". The sentence is positive for "panel" and negative for "remote" (mixed opinion).

The *aspect level analysis* focuses on identifying aspects that have been commented on, i.e. mining opinions from a text document about specific entities and their aspects, which can provide valuable insights to both consumers and businesses. Therefore, more robust and fine-grained evaluation of the opinions expressed for a particular topic is possible (Liu et al., 2010). In an ordinary opinionated text, an author might write about both positive and negative aspects of the object, even though the overall sentiment towards the object may be positive or negative. As document-level and sentence-level classification do not contribute that kind of information, the aspect-level classification is of particular interest.

The task can be divided into four steps (Pontiki et al., 2016):

- Step 1: *Aspect Term Extraction*

  The task is to determine the aspect terms mentioned in the text and return all the specific aspect terms. For example, in the sentence "The room in our hotel was beautiful", the aspect is the room of the hotel.

- Step 2: *Aspect Term Polarity*

  Identify the polarity of each aspect term that has been determined in step 1 (e.g. positive, neutral or negative).

- Step 3: *Aspect Category Detection*

  The aim is to determine the aspect categories discussed in a particular text. In comparison to the aspect terms in step 1, aspects categories are not as finely grained, and they do not have to appear as terms in the given text. In the analysed entity of hotels, the categories could consist of rooms, price, food and service.

- Step 4: *Aspect Category Polarity*

  After identifying the aspect categories used in a specific text in step 3, the aim is to determine the polarity (e.g. positive, negative or neutral) of every aspect category.

2.2 Approaches to sentiment analysis

The different technical approaches to sentiment analysis can be roughly divided into two areas (Collomb et al., 2014).

- *Lexical analysis (i.e. Lexical classifier)*
- *Machine learning analysis*

The *lexical analysis* determines sentiment polarity based on the semantic orientation of words or sentences in a given document. Therefore semantic orientation of an opinion on a particular feature f identifies if the opinion is positive, negative or neutral (Chong and Mastrogiovanni, 2011). It is commonly used in the absence of any labelled data. In the first step, the text of interest is tokenized, and a dictionary consisting of pre-tagged lexicons is used to match those tokens. A negative or positive match influences the total pool of score for the text. For instance "horrible" is a negative match in the dictionary. Therefore the total score of the text decreases. For a positive tagged word, the score is increased. Hence, the classification of a text depends on the overall score it achieves. The probability of a text T being positive can be computed as $P(+|T) = a + a/b$, where a and b are the number of positive and negative occurrences respectively (Melville et al., 2009). The overall sentiment of text T depends on a predefined threshold t. For $P(+|T) > t$ the text is classified as positive; otherwise, T is classified as negative. Without any prior knowledge about a threshold, the value of 0.5 can be found in the literature (Melville et al., 2009).

The machine learning approach has gained popularity over the last couple of years due to its accuracy and adaptability (Thakkar and Patel, 2015). It uses several learning algorithms to predict the sentiment based on a set of training data. Afterwards, a test dataset is used to evaluate the performance of the predictions. Machine learning approaches are most useful for sentiment classification of categorized text into positive, negative and neutral categories (Collomb et al., 2014). Supervised learning techniques, such as support vector machines (SVM) and Naive Bayes classifier are commonly used for sentiment prediction. In Chapter 3, Long short-term memory (LSTM) models are introduced in the context of artificial recurrent neural networks. The predictive performance will be compared to an SVM model. According to Pang et al. (2008), discriminative classifiers like SVM are superior concerning sentiment classification compared to generative models, because they distinguish between mixed sentiments (i.e., one document uses positive and negative words). In a small set of training data, the Naive Bayes classifier might be more suitable as SVM relies on a more extensive training dataset to build a high-quality classifier Sadegh et al. (2012). The polarity of each entry in the training data has to be labelled manually, which can be a time-consuming task. After creating a model based on the training dataset, it is used for classification on a previously unseen text. The model performance is evaluated by different indexes (e.g. Accuracy, F1 Score, Recall). In a simple classification model with only two categories(e.g. positive and negative), the indices are computed based on the following confusion matrix:

	Predicted Negative	Predicted Positive
Actual Negative	TN	FP
Actual Positive	FN	TP

Figure 1: Confusion Matrix based on a binary classification Problem (Chawla et al., 2002).

- TN (True negatives) is the number of negative observations that were classified correctly.
- FN (False negatives) is the number of negative observations falsely classified as positives.
- FP (False positives) is the number of positive observations falsely classified as negatives.
- TP (True positives) is the number of positive observations that were classified correctly.

A commonly used performance measure in the context of machine learning algorithms is predictive accuracy. It is defined in the following way (Chawla et al., 2002):

$$Accuracy = \frac{(TP+TN)}{(TP+FP+TN+FN)}$$

For a balanced dataset and the same error costs, the error rate $(1 - Accuracy)$ can be used as a performance metric (Chawla et al., 2002). When facing an imbalanced dataset and different error costs among the classes, other performance metrics like Recall and Precision should be used.

$$Recall = \frac{(TP)}{(TP+FN)}$$
$$Precision = \frac{(TP)}{(TP+FP)}$$

The recall summarises the capability to detect all relevant observations in a dataset. Precision, on the other hand, computes the proportion of instances a model declared as relevant that was relevant. An appropriate choice of performance metrics for the task at hand will be motivated in section 3.5.4. To conduct a sentence-level sentiment analysis for Coindesk (2019) news, different machine learning approaches are introduced in the next chapter.

3 Deep learning methods and algorithms

In the following chapter, the architecture of several neural networks is discussed. Advantages and disadvantages are outlined, and the LSTM model is introduced. Further, critical features of the network like word embeddings are explained, and the algorithm setup of the LSTM model for sentiment prediction is examined in detail. The training process of the model is covered with an emphasis on problems that came up during that process, followed by possible solutions. Moreover, support vector machines (SVM) are introduced. The chapter concludes with an overview of performance results for differently specified LSTM models and the SVM model covered by Bommes et al. (2019).

3.1 Deep Neural Networks (DNNs)

Deep Neural Networks (DNNs) was first introduced by Grossberg (1988) in the context of mathematical models of the information processing capabilities of biological brains. Nowadays the similarities between biological neurons and DNNs are proven to be rather weak. The popularity of DNNs is still rising in the field of pattern classification. They are used to solve complex problems like natural language understanding. In a neural net, the computer is not told how to solve a particular problem. Instead, it uses observational data to learn from, calculating its solution to the problem (Yin, 2017).

3.1.1 Architecture

The general architecture of a fully connected DNN can be described as a network of small processing units (often called nodes) that are linked to each other by weighted connections (Hinton et al., 2006). In the biological use case, each node represents a neuron, and the weighted connections serve as the strength of the synapses between the neurons. The network is initialized by feeding input to one or multiple nodes and advances throughout the hole network, passing the weighted connections generating an output (Kawakami, 2008).

The units in a multilayer fully connected DNN are organized in layers. Each connection is bringing information forward from one layer to the next. In general, a fully connected DNN consists of at least three layer types: the input layer, the hidden layer and the output layer (Yin, 2017). Each unit of the input layer represents a feature x_i of the input data, and each unit of the output layer represents at least one class y_i (Yin, 2017). The number of layers defines the depth of a model; the maximum size of the layers defines the width of the model. By adding more hidden layers, the DNN can describe highly complex functions.

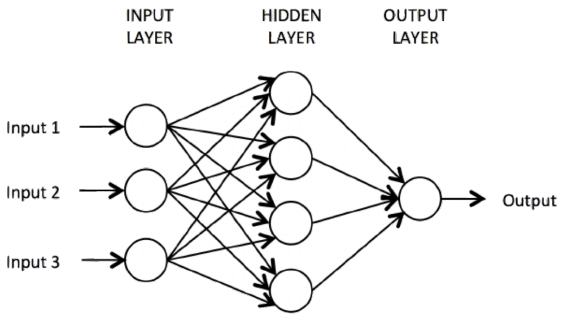


Figure 2: Fully connected neural network. Three Input layers, one output layer and four hidden layers (Kim, 2014).

The forward pass of a network describes the process of the resulting unit activations form the input layer getting disseminated through the hidden layers to the output layer (Yin, 2017). There exist different activation functions (mostly nonlinear) for the units in the hidden layers, which transform the summed activation arriving at the unit (e.g. sigmoid/logistic activation function, rectified linear unit (ReLU) and softmax function). Selecting an activation function is an essential task as it can affect the way that input data has to be formatted. The choice of a proper activation function will be looked at in detail later on.

A fully connected DNN, as shown in Figure 2 has several advantages (Yin, 2017):

- *Less need for engineered features*: When solving Natural language processing (NLP) problems, one huge task is feature engineering. The conventional way was creating features manually. DNN's learn task-specific features from the training data, implementing feature engineering automatically.

- *Parameterize all system components*: The input X and the output Y are known and fixed in the shallow machine learning systems. For DNNs, all parts, including inputs, connections and outputs, are parameterized.

- *Reliable generalization power*: DNN‘s have a vast number of trainable parameters that can extend the amount of training samples. According to Zhang et al. (2016), the

generalization error of these models, i.e. difference between "training error" and "test error is still quite small. This is largely due to the fact that inputs are represented in both training dataset and testing dataset. Therefore parameterized continuous representations create connections between samples, that stand far away from each other in conventional representation schemes.

3.1.2 Convolution Neural Networks (CNN)

Convolutional neural networks (CNNs) apply layers with convolving filters that are enforced on local features (LeCun et al., 1998). They are useful for NLP tasks, especially for semantic parsing (Yang et al., 2014), search query retrieval (Shen et al., 2014) and sentence modelling (Kalchbrenner et al., 2014).
The basic idea behind CNNs is to reduce the connections between input and hidden layers instead of fully connecting them (LeCun et al., 1998). Although some of the benefits of a fully connected network were given above, the downside is the vast amount of parameters that need to be trained, which leads to large parameter matrices and the matrix multiplications are computationally expensive.

A CNN model consists of convolution, pooling, fully connected, and an additional dropout layer (Kim, 2014).

Convolution layer: To obtain more refined data, this layer applies convolution operations on the input data by using a kernel, or fixed sized filter.

Pooling layer: At this stage, the different vectors that were outputted by the convolution layer are pooled, with only the most relevant vectors being passed forward.

Fully connected layer: CNNs have at least one fully connected layer after the Convolution layer. It looks at the output of the previous layer and determines which features most correlate to a particular class. Those will be part of the output.

Dropout layer: The dropout layer is an optional layer to prevent overfitting. Therefore, during the training process, a probability p is used to randomly prevent different weights from updating.

Since the output of a feedforward neural network (FFNN) or multilayer perception, like the ones above, only depend on the current input, and not on the past or future contributions, such networks are more applicable for pattern classification tasks (Specht, 1990).
When dealing with text input, the input size often varies, and the exact amount of input in streaming data is hard to determine. Therefore a different neural network is introduced in the next section that makes it possible to share parameters at different time steps.

3.2 Recurrent neural networks (RNNs)

The architecture of an RNN varies from the FFNNs that were introduced above. We can think of the RNN as an extension of the CNN. RNNs include cycles that pass the activations from previous periods as input to the network. Based on previous periods, the network will decide on the current input (Sak et al., 2014). When analyzing text, the remote meaning of a word cannot be given, only the context of a sentence (Janssen, 2001). The meaning of a longer expression (e.g. word in a sentence) also depends on the meanings of its surroundings (Tang et al., 2016). Recurrent Neural Networks can deal with short term dependencies in sequential data (Lin et al., 1996). The part of the network which is responsible for including previous inputs is called recurrent hidden state. According to Sak et al. (2014), for a sequence of inputs $x = (x_1, x_2, \ldots, x_T)$ the recurrent hidden state h_t is updated by

$$h_t = \begin{cases} 0 & t = 0 \\ \phi(h_{t-1}, x_t) & \text{otherwise} \end{cases} \tag{1}$$

Where ϕ is a nonlinear function.
The output of the RNN can be notated as $y = (y_1, y_2, \ldots, y_T)$. The recurrent hidden state h_t in 1 is traditionally implemented in the following form:

$$h_t = \sigma(Wx_t + Uh_{t-1} + b) \tag{2}$$

with σ being a bounded and smooth function (e.g. hyperbolic tangent function, sigmoid function), W and U are weighting parameters to regulate the input and b is a bias. The hidden state h_t is a function of all previous hidden states, which shows that RNNs are inherently deep in time (Salehinejad et al., 2017).
The sigmoid activation function is notated in 3.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3}$$

Figure 3 illustrates the hidden state equation 2. The grey boxes are layers with an activation function like the sigmoid activation function notated in equation 3.
The output is a sequence of length T. At every time interval t, an input x_t is handed to the model, and the output h_t is generated, which acts as an input to the model at the next time interval.

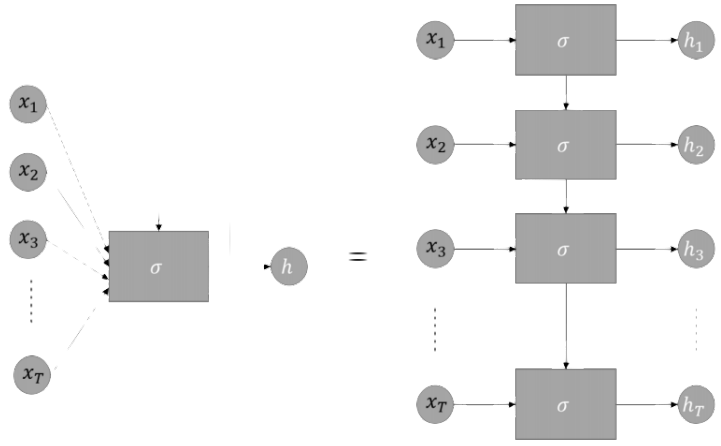


Figure 3: Illustration of the hidden layer in a Recurrent neural network (Salehinejad et al., 2017).

In natural language, a text tends to have long term dependencies. For example, when trying to predict the next word in the sentence:
"I spent three years in Argentina. .... I speak fluent (...).".
Obviously, the next word in the sentence should be *Spanish*. Based solely on recent information, an RNN would predict the next word to be the name of a language. To conclude, which language is correct, previous parts of the text have to be included as well. Mikolov et al. (2010) address these shortcomings of RNNs when dealing with long-term dependencies. It is challenging to train RNNs to capture long-term dependencies, as gradients either vanish or explode, which causes problems while using the gradient-based optimization method (Wang et al., 2016). This is based not only on the variations in gradient magnitudes but on the long-term dependencies being hidden by the effect of short-term dependencies as well (Chung et al., 2014). These long-term dependencies can have a strong influence on the overall polarity and meaning of a document. One way of dealing with this issue is the introduction of the Long Short Term Memory network.

3.3 Long Short-Term Memory (LSTM)

For tasks that require capturing long-term dependencies, (e.g. speech recognition(Graves et al., 2013) and machine translation (Luong et al., 2014)) RNNs using recurrent units have gained popularity over the past years. One of them is the Long Short-Term Memory(LSTM) unit, which was first introduced by Hochreiter and Schmidhuber (1997). The other one is called gated recurrent unit and was introduced more recently by Cho et al. (2014). Although this thesis will focus on the former, both of them are used in practice.

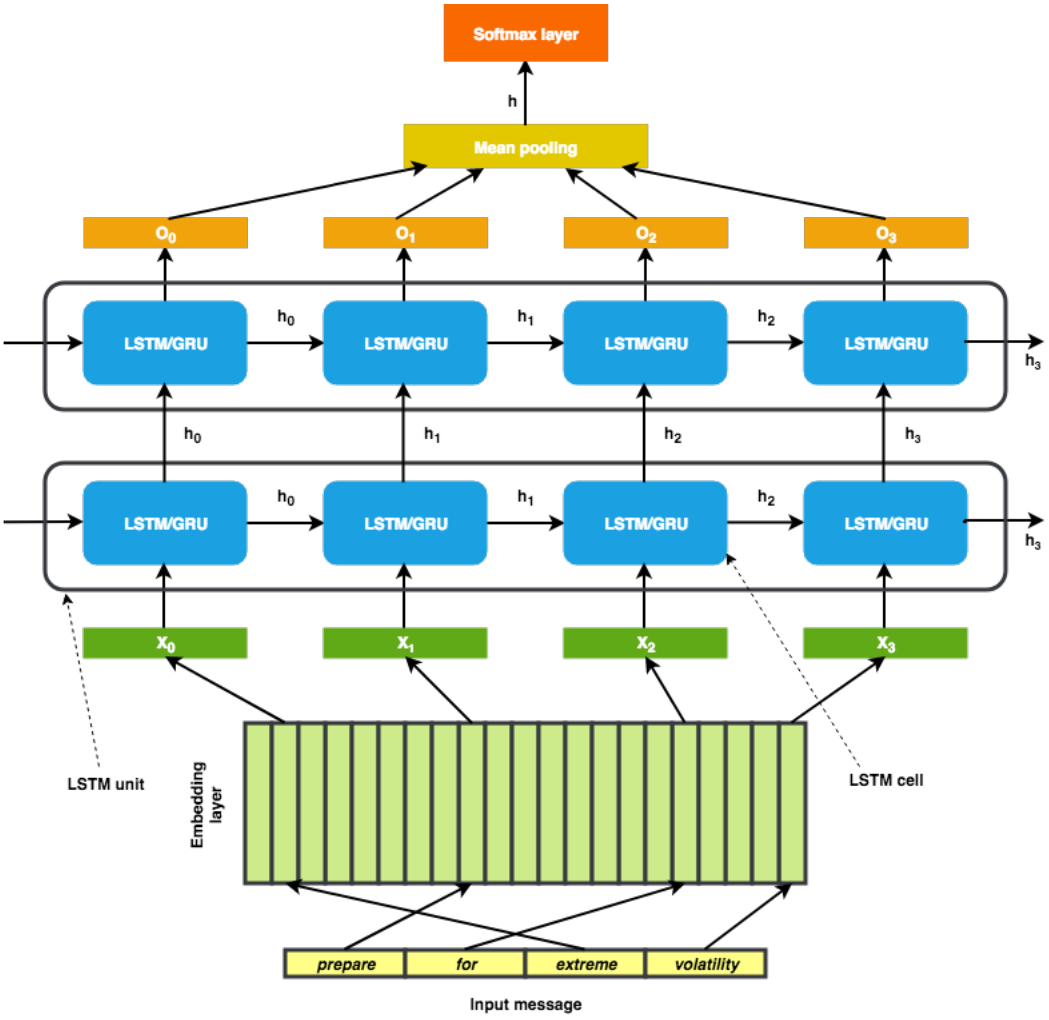


Figure 4: General Architecture of an RNN Nasekin and Chen (2018).

Figure 4 shows the full architecture of an RNN with multiple LSTM/GRU gates. It is composed of the input sequence, an embedding lookup matrix, several layers of LSTM/GRU cells, an output sequence, mean pooling and softmax layers.

The main components of the RNN are the LSTM/GRU cells. A structural example can be found in Figure 5, and Figure 6, respectively. One additional feature of the LSTM architecture in Figure 4 is the cell states C_t, which gives the network the possibility to store information about previous states of LSTM cells. What information is stored in a particular cell state is guarded by the gates. There are three different gates: an *input gate* i_t, a *forget gate* f_t and an *output gate* q_t. In the first step, the *forget gate* f_t regulates, how much of the former state C_{t-1} will flow into C_t, related to the values of the past hidden state h_{t-1} and the current input x_t (Nasekin and Chen, 2018). The *forget gate* f_t is similar to the hidden layer introduced above and can be formally notated as:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{4}$$

with the sigmoid function $\sigma(x)$ generating output for each number in the cell state C_{t-1} between 0 and 1:

$$\sigma(x) = \frac{1}{1 + exp(-x)} \tag{5}$$

In the next step, the LSTM cell conducts an update to C_{t-1} by computing a new candidate value of the cell state, $\tilde{C}_t$ with a tanh layer:

$$\tilde{C}_t = tanh(W_c x_t + U_c h_{t-1} + b_c) \tag{6}$$

where $tanh(x) = \{exp(x) - exp(-x)\}/\{exp(x) + exp(-x)\}$.

The input gate i_t plays a vital role to determine what will be stored in the next cell state C_t. It controls the amount of information from the new candidate state $\tilde{C}_t$ that will be inputted into C_t. Similar to the forget gate from the previous step i_t generates numbers between 0 and 1 for every value of $\tilde{C}_t$:

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{7}$$

The cell state in the current period C_t is a weighted sum consisting of the past cell state C_{t-1} and the new candidate state $\tilde{C}_t$:

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \tag{8}$$

with $\odot$ denoting an element-wise multiplication.

Lastly, an updated value of the hidden state in period t, h_t can be calculated. Therefore the cell state C_t is used in the tanh function and multiplied element-wise by the output gate g_t:

$$h_t = g_t \odot tanh(C_t) \tag{9}$$

with $g_t = \sigma(W_g x_t + U_g h_{t-1} + b_g)$. The hidden state value h_t is then disseminated within LSTM units, between units, across LSTM cells, and also upwards to the next hidden layer. One important distinction to make at this stage is the one between LSTM cells and units. The former is illustrated in Figure 4 (blue boxes) and is described by the equations above. The latter is a group of LSTM cells, which is highlighted as the black box in Figure 4. Moreover, Figure 5 gives an in-depth look into the architecture of such an LSTM unit (Nasekin and Chen, 2018).

Each LSTM unit generates an output sequence $h_0, h_1, h_2, \ldots, h_n$, which serves as the input for the next unit as well as for the next layer. This specific kind of architecture grants the LSTM network to model long term dependencies efficiently, as the cell state of previous periods influences the ones in future periods as well as next layers. The name "long short-term memory" is derived from the networks ability to balance "old" and "new" information using recent input events. Another major advantage of this feature is the reduced problem of vanishing (or exploding) gradients that has been addressed above.

Another recurrent unit that is used in the context of RNNs is the GRU (Gated Recurrent Unit). Although it is not featured in the next parts of this thesis, the basic architecture and functionality are shortly outlined. A similar set of equations can notate the GRU, and its architecture is illustrated in Figure 6. It consists of an update gate z_t, which is a combination of the input and forget gate in the LSTM unit, and a reset gate r_t. Besides the cell state and the hidden state are combined into one state h_t. The update gate z_t combines the information provided by the previous hidden state h_{t-1} and the new candidate value $\tilde{h_t}$, by building a weighted average of the previous hidden state h_{t-1} and the new candidate value $\tilde{h_t}$. The reset gate r_t establishes the amount of past information from h_{t-1} that will be forgotten. More formally the GRU with its features can be described by the following formulas (Nasekin and Chen, 2018):

$$z_t = \sigma\left(W_z x_t + U_z h_{t-1}\right), \tag{10}$$

$$r_t = \sigma\left(W_r x_t + U_r h_{t-1}\right), \tag{11}$$

$$\tilde{h}_t = \tanh\left(W_h x_t + r_t \odot U_h h_{t-1}\right), \tag{12}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t. \tag{13}$$

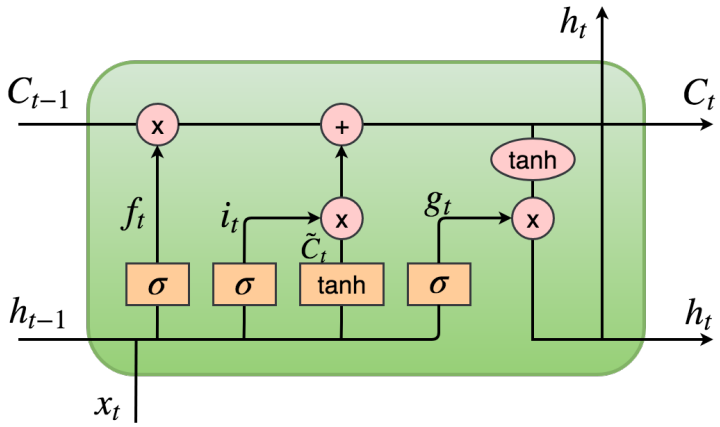


Figure 5: Structure of an LSTM unit Nasekin and Chen (2018).

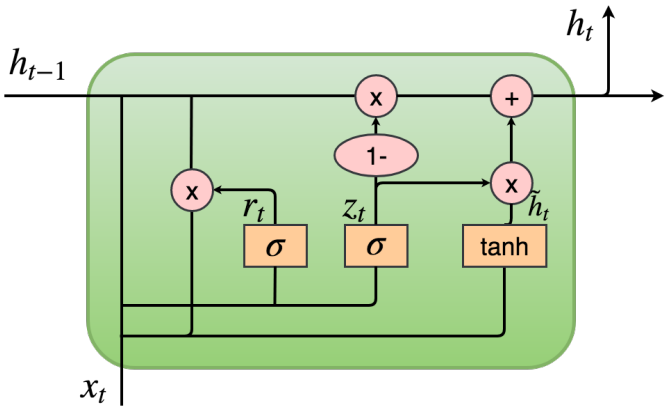


Figure 6: Structure of an GRU unit Nasekin and Chen (2018).

Word embeddings

An accurate word representation is crucial for natural language processing. Often words are represented as discrete and distinct symbols, which is scarce for many tasks and lacks the ability of generalization (Levy and Goldberg, 2014). For example, the symbolic representation of the words "club mate" and "coffee" are entirely unconnected. Even though "coffee" is a valid indication of the verb drink, there is no information that it is also an indicator for "club mate". To successfully train a statistical model based on the symbolic representation, more data might be needed. Therefore the representation of words has to capture semantic and syntactic similarities between them (Alharbi, 2016). A distributed representation of words, defined as real-valued, dense, and low dimensional vectors, is called word embedding (Liu et al., 2015). The syntactic or semantic properties of a word are thereby described by each dimension of the word embedding vector (Zhao and Zhao, 2019).

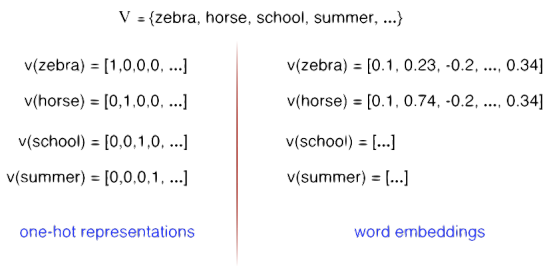


Figure 7: Comparison between one hot encoding and word embeddings Yin (2017).

In Figure 7 the difference between one hot encoding and word embeddings is shown. In one hot representation, for a given vocabulary with size V, every word is denoted as a binary vector of length V with a value of 1 at the word specific index and 0 for the remaining values. The approach of one hot encoding is used in practice due to its simplicity but is suffering from memory inefficiency and the inability to detect word similarities (Yin, 2017). According to the left-hand side of Figure 7, none of the potential pairs out of the four words share any similarity. The distance between two words is the same for every possible combination. However, it is apparent that "zebra" and "horse" should be more similar than "zebra" and "school" as they are pretty analogous animals. A word embedding model using Word2vec technique, e.g. right-hand side in Figure 7, can represent the distance of a particular word from the rest of the dictionary via a form of distribution relationship, also known as distributed Representation (Nguyen et al., 2012).

As indicated in Figure 7, a single word in the embedding space is represented by a d-dimensional vector (with mostly $d << V$) (Yin, 2017). Its main benefit is the ability to share information between similar words. Such related words, e.g. "zebra" and "horse" in the example above, will have similar vector values. As in Figure 7, the two candidate words share the same feature values besides the second value. The usage of such low-dimensional and dense vectors has two main benefits. One is computational, and the other is its generalization power. Once, similarities between features are detected, it is worthwhile to find a representation that can secure these.

For example, during training, the word "horse" has been observed numerous times, but the word "zebra" only a couple of times. Each word is identified with its dimensions; instances of "horse" do not carry information about the ones of "zebra". In dense vector representation, the vector for "horse" might share some properties with the vector of "zebra", enable the possibility to share statistical information between the two (Yin, 2017). According to Nasekin and Chen (2018), a pre-trained matrix embedding leads to faster and smoother convergence for the training algorithm. Different methods for pre-trained word embedding weights have been introduced in the literature (e.g. Word2Vec and GloVe).

The most popular family of methods is the Word2Vec model. Two methods for generating dense embeddings are looked at in particular: skip gram and continuous bag of words (CBOW).

Both act as each other's counterpart. For the skip gram method, the context words $c1, c2, \cdot, cC$ are predicted for a given word w. For CBOW, a word w is predicted based on the context words $c1, c2, \cdot, cC$. So in a slightly different notation, the goal of a skip-gram model is maximizing the conditional probability $p(c|w; \Theta)$. Hence choosing a context word that is the most likely under the condition of observing a given word w. This probability can be represented as a softmax function (Nasekin and Chen, 2018):

$$p(c|w;\Theta) = \frac{e^{v_c \cdot v_w}}{\sum_{\substack{c' \in C \\ c' \notin c}} e^{v'_c \cdot v_w}} \tag{14}$$

where the vector embeddings of c and w are notated as v_c and $v_w \in R_d$. The parameter Θ from the conditional probability is the vocabulary and the set of all contexts, P and C. In particular v_{c_i}, v_{w_i} for $w \in P$, $c \in C$.

Therefore the objective function that has to be maximized can be notated in the following form:

$$\underset{\Theta}{\operatorname{argmax}} \prod_{W \in P} \prod_{c \in C} p(c|w;\Theta) \tag{15}$$

The task carried out in equation 15 has one major downside. The summation over all c' in the denominator in 14 is very computationally expensive and therefore not practical. One way to overcome this problem is by using negative sampling, which chooses different "noise" words from the corpus by virtues of their frequency.

Additionally to the "normal" pair $(w, c) \in D$, a second pair, the "noise" pair $(w, c) \in D'$ is generated. Note that $D \cup D'$ constructs the whole corpus (Nasekin and Chen, 2018). For the notation above the negative sampling objective can be conducted in the following way:

$$\underset{\Theta}{\operatorname{argmax}} = \sum_{(w,c) \in D} log\sigma(v_c \cdot v_w) + \sum_{(w,c) \in D'} log\sigma(-v_c \cdot v_w) \tag{16}$$

witch σ as softmax function. A more detailed look into the computation of the negative sampling approach can be found at Levy and Goldberg (2014).

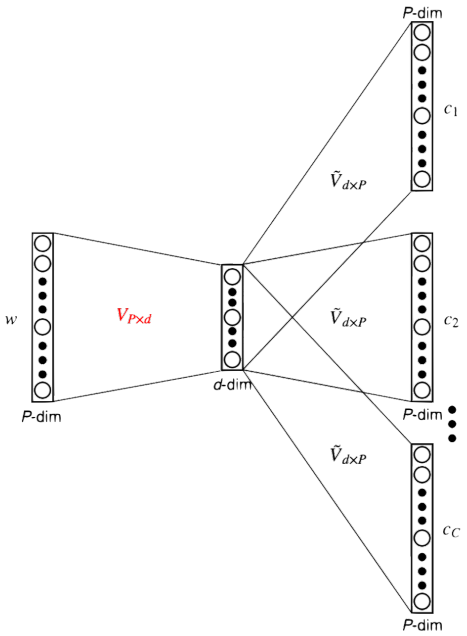


Figure 8: Architecture of a Word2Vec neural model (Nasekin and Chen, 2018).

A Word2Vec model is illustrated in Figure 8. As the Graph shows, it is a shallow neural network with one hidden layer, containing shared weights $\tilde{V}$ for all context words c. One downside to the method of negative sampling in 16 is its lack of optimal predictions for context words, as it is an approximation of 15.
Nonetheless, it provides meaningful embeddings V, which can be used to train an RNN model. Another benefit of the Word2Vec model is the dense vector representation for vocabulary words. The vectors dimensions (d) can be much smaller compared to the size of the dictionary P (Nasekin and Chen, 2018).

3.4 Support Vector Machines (SVM)

A different machine learning classification technique is support vector machines (SVM). It can be used for sentiment classification as well as different types of NLP. It is an extension of the support vector classifier and uses a kernel function to map data points from a space in which the data is not linearly separable into a new space where it is (Gelbukh, 2006). Separating the data points is done by using non-linear boundaries. For a more detailed introduction to the topic, please see Joachims (2002). When working with a multi-class classification problem, SVM is not able to classify all observations directly, but by a split into binary tasks (Joachims, 2002). For every class i, a binary classification problem can be set up in the following way (Joachims, 2002):
The class label is ${y_{bin}}^{(i)} = +1$ if $y = i$, for the i-th binary learning task. Accordingly, the binary class label is ${y_{bin}}^{(i)} = -1$ if $y \neq i$.
An independent and identical distributed (i.i.d) training sample S of n examples

$$(\overrightarrow{x_1}, y_1), \ldots, (\overrightarrow{x_n}, y_n) \tag{17}$$

is drawn, with the document vector $\overrightarrow{x_1}$ and class label y. The document vector $\overrightarrow{x}$ is a high dimensional vector carrying information about the words in the document.
Afterwards, each binary problem is used to train an individual classifier on, which results in l binary classification rules $h^{(1)} \ldots h^{(l)}$. For the classification of a new example $\overrightarrow{x}$, the output of the estimate of $Pr(y = i|\overrightarrow{x})$ is inspected. It is classified as an instance of the class for which $Pr(y = i|\overrightarrow{x})$ is the largest. Subdividing multi-classification problems into l binary classification problems is often called $one - versus - all$(OVA) strategy (Joachims, 2002). A different, but less frequently used approach is pairwise classification. Applying this strategy leads to $l(l-1)/2$ classification problems.

Bommes et al. (2019) implemented an SVM model based on the OVA approach to predict the polarity of each sentence in the Malo et al. (2014) dataset. The next section will compare the performance of the SVM implemented by Bommes et al. (2019) and the LSTM model introduced in this thesis.

3.5 LSTM method applied to sentiment prediction

3.5.1 Algorithm setup for sentiment prediction

The framework of the LSTM Model used to predict sentiment for the Coindesk dataset, is based on research carried out by Peter Nagy (2018). The aim of his analysis differs slightly from the study conducted in this thesis, as the author's goal is to predict sentiment for Twitter posts, in which only positive and negative polarity is considered. The language used in twitter posts is informal and can include slang. At this point, it is essential to remember that differences in style and structure of training and prediction datasets influence the results of the analysis. As the model is fitted to a specific type of language, it may not be able to replicate the results on unseen data that is based on a different style or form of language. The base architecture introduced by Peter Nagy (2018) is still useful for this thesis since the training process of the model is the crucial component to gather information subjected to language structure. However, the structure of the model needs to be specified correctly, depending on the task at hand. Hence some changes to the model will be made. Table S1 gives an overview of the Parameter-setup for the LSTM used by Peter Nagy (2018) (left-hand side). On the right, the model specifications for sentiment prediction of three categories (positive, neutral and negative), based on the Malo et al. (2014) training data is notated. Throughout this chapter, the reasoning behind the choice of different parameters will be made as transparent as possible. The field of deep learning is rather young. Therefore it might not be possible to identify a state of the art technique for every task at hand.

Maximum encoded message length (*input_length*)	28	57
Embedding dimension (*embed_dim*)	128	32-512
Batch size	32	32
SpatialDropout1D	40%	40%
Recurrent Units (*lstm_out*)	196	196
Recurrent dropout	20%	40%
Dropout	20%	40%
Recurrent layers	2	3
Activation function	softmax	softmax
Loss function	Binary Cross-Entropy	categorical crossentropy
Optimizer	Adam	Adam

Table S1: Parameter-setup for LSTM by Peter Nagy (2018) on the left side. Adjusted Patameter-setup on the right side.

Input Length

The networks *input_length* depends on the maximum vocabulary size of each sentence in the training data (Pennington et al., 2014). Therefore the range of each statement in the Malo et al. (2014) data is analysed in figure 9. The mean length of a sentence is 22.45 words. There are only a few sentences with a length of 60 and above. One could check manually if those sentences carry any additional information or whether they can be excluded from the analysis. The most prolonged instances do not provide new insights, mainly a listing of names and nationalities [1]. Therefore the *input_length* is fixed at 57. By excluding long sentences, the density vector (*input_length*) is shorter, and in deeper neural networks, this can provide computational benefits. Note that the analysis of word frequency has been carried out on the dataset with a 100% agreement rate. The optimal *input_length* differs with different agreement rates.

[1] For instance: "Supported Nokia phones include : N96 , N95-8GB , N95 , N93-N931 , N92 , N85 , N82 , N81 , N80 , N79 , N78 , N77 , N76 , N75 , N73 , N72 , N71 , E90 , E71 , E70 , E66 , E65 , E62 , E61-E61i , E60 , E51 , E50 , Touch Xpress 5800 , 6220 Classic , 6210 Navigator , 6120 Classic , 6110 Navigator , 5700 , 5500 , 5320XM."

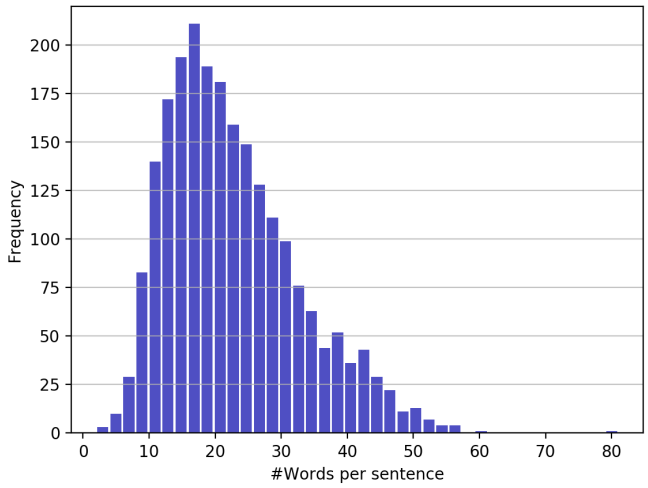


Figure 9: Word Frequency in Malo et al. (2014) training data for a 100 % agreement rate amongst annotators.

Embedding Dimensions

Afterwards, the embedding layer translates the *input_length* into a sequence of dense vectors with the dimensions of *embed_dim*. According to Britz et al. (2017), larger embeddings are supposed to generate higher BLEU(Bilingual Evaluation Understudy) scores. Additionally, they are expected to result in a lower perplexity of the model.

Although 2048-dimensional embeddings generate the best results, they only do so by a small surplus. Even way smaller embeddings, such as the 128-dimensional embedding achieved quality results and converged almost twice as fast.

Furthermore, Britz et al. (2017) observed that gradient updates to small and large embeddings had no significant difference and regardless of size, the norm of gradient updates to the embedding matrix stayed roughly constant while training. No overfitting with a larger embedding size was observed, and the number of trainable parameters increases rapidly with the size of the embedding dimensions, while the training log perplexity stayed approximately the same. Britz et al. (2017) suggest that the model does not make sufficient use of the additional parameters and that better optimization techniques might be needed. Although higher embedding dimensions yield better results, they only do so by a slight margin, and in case computational power is sparse, a smaller dimensionality might be beneficial. For the given reasons above the embedding dimensions are set to 512.

Batch size

The Batch size determines the number of samples per gradient update. Each Batch results in an update to the model. Accordingly, a smaller batch size contributes to a faster learning rate (Keskar et al., 2016). For many NLP deep learning tasks, the stochastic gradient descent method has been the algorithm of choice. It only uses a small portion of the training data, generally between 32 and 512 data points to conduct an approximation of the gradient. Keskar et al. (2016) have further shown that a larger batch size leads to a significant degradation for the quality of the model, which manifests in a weaker ability to generalize. According to Keskar et al. (2016), this may be based on a particular property of large-batch methods, as they tend to converge to sharp minimizers (of the training function). By comparing the model results based on different batch sizes, between 32 and 512, an optimal batch size of 64 could be identified.

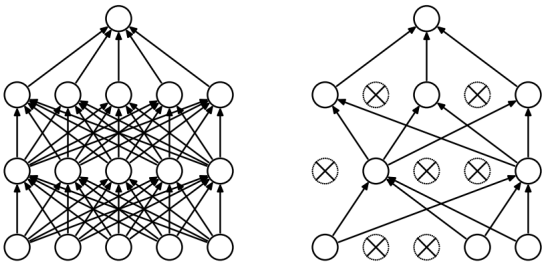
Dropout

One significant difficulty when working with Neural Networks is their habit of overfitting on training data. As NNs consist of multiple non-linear hidden layers, they can learn extremely complicated relationships along with their in and outputs. However, sampling noise might be the reason for a lot of these relationships and will, therefore, exist only in the training data, but not in the data that is used for prediction (Srivastava et al., 2014). Many methods have been developed to attenuate this issue. One way to prevent overfitting[2] is by implementing a dropout, which refers to dropping units in a recurrent neural network. The dropped unit is temporarily removed from the system, including all its incoming and outcoming connections (Deng et al., 2014).

An individual unit is cut from the network with a probability p, independent from other units (Srivastava et al., 2014). As shown in Kim (2014), a dropout probability p of 0.5 is optimal for most scenarios. One can think of it as sampling a "thinned" neural network from the original one. All units that are unaffected from the dropout constitute the "thinned" network. Therefore a network with n units can be transformed into 2^n possible "thinned" networks. The process of training a neural network with dropout is similar to training an assemblage of 2^n thinned networks (Srivastava et al., 2014).

Although dropout is a powerful tool for feedforward neural networks, it lacks the same abilities for Recurrent neural networks such as LSTM. According to Bayer et al. (2013), the lack

[2]The concept of overfitting will be further introduced in Section 3.5.4



(a) Standard Neural Net (b) After applying dropout.

Figure 10: Left: Standard NN with two hidden layers. Right: Thinned net produced by applying dropout. Crossed units have been dropped (Srivastava et al., 2014).

of performance is due to the amplified noise produced by the recurrence. As a result, the models learning ability to store information over several periods is limited. A possible solution to this problem is presented by Zaremba et al. (2014). They are applying the dropout only to those connections that are non-recurrent. Therefore the LSTM can improve from dropout without losing its ability to memorize instances from past periods.
As the dropout percentage suggested by Peter Nagy (2018) is relatively low compared to the optimal dropout percentages presented in the literature, a dropout rate of 0.4 and a recurrent dropout rate of 0.4 is used when compiling the model. This means that 40% of the units for the linear transformation of inputs are dropped. Subsequently, the linear transformation of the recurrent state drops 40% of units as well.

Recurrent layers

The recurrent layers in an LSTM are part of the dense layer, connecting each layer to all subsequent layers. Huang et al. (2017) have discovered a regularizing effect of dense connections, reducing overfitting on a task with only limited training data. As the output of the LSTM is constructed to take on three different values (positive, neutral and negative), the recurrent layer has to create three output values, one for each class. The model introduced by Peter Nagy (2018) focuses on a more straightforward task. The sentiment is only classified into two categories, negative and positive.
The task of sentiment classification has been defined as a two-category problem by many researchers (e.g. Pang and Lee (2005) and Dave et al. (2003)). However, not every comment of a user, or article in the news can be declared as a positive or negative statement.

In most cases, they consist of facts towards a specific topic, without carrying any sentiment. Therefore the third class of neutral sentiment is necessary. Koppel and Schler (2006) explored the thesis that neutral documents show less potential to learn from than materials with a clearly defined sentiment. They concluded that learning only from positive and negative examples will lead to a misclassification of neutral cases.

Further, the use of neutral sentiment improves the distinction between positive and negative examples. Especially when using the training data from Malo et al. (2014), neutral sentiment plays an important role. As discussed above, discriminating between neutral and positive sentiment achieved the lowest pairwise agreement rate amongst annotators (74.9%). Hence, excluding neutral sentiment from the analysis may overestimate the proportion of positive instances, as the distinction between the two is somewhat hard in the context of financial news. Although a three-category task is more complex, the additional information gained by the inclusion of a neutral class makes an implementation mandatory. For the given reason, the model consists of three recurrent layers.

3.5.2 Model Training

Pretrained word embeddings

The model consists of 754.039 parameters, from which 233.439 are trainable and 520.600 non-trainable. During training, individual weights are not updated. Constant weights throughout the training process are referred to as non-trainable parameters. The rather large number of constant weights in the model above can be explained by the use of pre-trained word embeddings (Global Vectors [3].).

The critical question regarding word embeddings is how meaning is generated from word occurrences and how this meaning is represented by the derived word vectors (Pennington et al., 2014). Particular features of meaning can be obtained by co-occurrence probabilities. Pennington et al. (2014) illustrate this idea by introducing a small example. Two individual words, i and j, carry a particular form of interest. In the context of thermodynamic phase, those words might be i = ice and j = steam. The relationship between the two can be inspected by looking at the ratio of their co-occurrence probabilities with different probe words, k. If a word k is linked to steam but not ice, for example, k = vaporous, the ratio of co-occurrence probabilities $\frac{P_j k}{P_i k}$ is expected to be large.

[3]Data can be found under : https://nlp.stanford.edu/projects/glove/

On the other hand, words k linked to ice but not steam, for example k = solid, the ratio is expected to be small. For words k related to both ice and steam or none of them, the rate is close to one. In the next step, the information present in the ratio $\frac{P_j k}{P_i k}$ has to be transformed into word vector space. For a more detailed mathematical approach to this transformation, please see Pennington et al. (2014). The resulting model constructs a vector space with purposeful substructure and combines the advantages of the two prevalent models in the literature, global matrix factorization and local context window methods. Finally, the GloVe corpora consist of 400.000 words with the associated pre-trained word embeddings. Later on in this chapter, the performances of differently specified models will be compared. One focus will be the differences in performance between a model using trainable word embeddings and one with word representations based on unsupervised learning (Word2Vec).

Loss function

A DNN is trained by the gradient descent optimization algorithm. As part of this, a loss function (often called error function) has to be defined, calculating the loss of the model for the current state. The results are used to update weights in the next evaluation to reduce the loss (Bishop et al., 1995). The choice of the loss function is an important task, as it has to correspond to the predictive modelling problem at hand. In this case, the loss function has to match a classification problem. The primary loss function used in the literature for classification problems is the cross-entropy function (Bishop et al., 1995). It will calculate a score, consisting of the average difference between the predicted probability distribution and the actual one for each particular class in the problem. The score needs to be minimized, with an optimal cross-entropy score of 0. Hence, the classification problem has more than two target values; an implementation of $categorical\_cross - entropy$ is needed. In a binary classification problem as seen in Peter Nagy (2018) $binary\_cross - entropy$ is used. As described above, the choice of output layers (Recurent layers) has to match the chosen loss function as well, which results in three recurrent layers for the $categorical\_cross - entropy$ loss function.

Activation function

An activation function is used to define the output of a NN. In this specific case, the output is one of the three sentiment classes (positive, neutral and negative). Generally, one distinguishes between two groups of activation functions: Linear and non-linear activation

functions. Hence, it is not relevant for the task at hand; the linear activation function will not be discussed. Due to its characteristics, the output of the linear activation function will not lay in any range, and therefore, it will not help with the complexity of different parameters. The benefits of a non-linear activation function lie in the ability to make it easier for a model to generalize and differentiate between the output. Four different functions are frequently used in practice (Bishop et al., 1995) :

- ReLU (Rectified Linear Unit) $R(a) = max(0, a)$:
  When using a ReLU activation function, the model's ability to train on given data is highly dependent on the data provided. Since its range is $[0, \infty)$, it cannot map negative values accordingly.

- Sigmoid or Logistic Activation Function $g(a) = \frac{1}{1+exp(-a)}$
  The sigmoid activation function has some beneficial properties. All resulting values lay in the range [0,1]. Therefore it is mainly used to estimate probabilities as model outputs.

- Tanh or hyperbolic tangent $g(a) \equiv tanh(a) \equiv \frac{exp(a)-exp(-a)}{exp(a)+exp(-a)}$
  Tanh is similar to the Logistic activation. It only differs through a linear transformation. Its values lay between [-1,1] and is therefore often used for binary classification problems. As Bishop et al. (1995) point out, "Tanh" can obtain faster convergence of training algorithms compared to Logistic functions. For that reason, it is often used instead of the Logistic function.

- Softmax $g(a) = {a_k}^* = \frac{exp(a_k)}{\sum_{k_1} exp(a_{k_1})}$
  As one can see, it shares properties with the sigmoid activation function, in terms of its values lying between [0,1]. It ensures that $\sum {a_k}^* = 1$, which enables the use of the cross-entropy error function and is beneficial in a 1 of q classification problem (with $q > 2$). Furthermore, Dunne and Campbell (1997) demonstrate how the logistic activation function can be rebuild, as a special form of the softmax. Due to its greater flexibility, softmax activation functions are widely used for multiclass prediction problems (Dunne and Campbell, 1997). The output is several probabilities, depending on how many class labels exist in the data. In the case of a three-way classification problem, each observation carries three probabilities, one for each class. The class with the highest probability is, according to the model, the most likely to be observed, given those particular characteristics (Tomas Mikolov, 2015).

Optimizer

The most popular and widely used optimizer algorithm for NLP tasks is Adam (Kingma and Ba, 2014). It internally tunes the learning rate for each parameter. A different optimizer that allows for more fine-tuning of the learning rate is the stochastic gradient descent (SGD). Although Adam converges faster, it has been shown that with some tuning of the learning rate SGD can outperform Adam (Zhang and Wallace, 2015). Most of the time, the choice of an optimizer is motivated rather poorly in the literature. As less tuning of the hyperparameters is needed, and Adam is implemented in the literature for NLP tasks, the further analysis will be based on models using the Adam optimizer.

3.5.3 Training Data

One major setback when training a model for textual analysis tasks on financial articles is the difference in vocabulary and expressions used by the media to report company related news articles (Loughran and McDonald, 2011). The availability of corporas in economic and financial domains is extremely sparse, especially for data sets that include a phrase-level annotation in the context of financial areas. Malo et al. (2014) built a corpus to train sentiment models for economic texts. They used English news articles on OMX Helsinki companies across different industries and news sources and were able to scrape 53.400 sentences from news articles. In the next step, a random sample of 5000 sentences was pulled and classified on a phrase-level into positive, negative and neutral categories. This task was carried out by three researchers and 13 master students from Aalto University School of Business. Each sentence of the corpus was analysed by five to eight participants. On a smaller subsample of 150 sentences, Malo et al. (2014) examined the degree of the overall pairwise agreement between all 16 annotators. High agreement rates were observed for separating between positive and negative sentences (98.7%), as well as between neutral and negative sentences (94.2%). The main challenge lies in the discrimination between positive and neutral sentiment, which manifests in a low pairwise agreement rate of 74,9 %. According to Malo et al. (2014), those findings might not be surprising, as companies tend to exaggerate during positive developments. A more detailed overview can be found in the Appendix in Figure S10.

Based on the number of overlapping annotations, Malo et al. (2014) provide four different ways of defining a "majority-vote-based gold standard" :

- *Sentences with 100 % agreement*
- *Sentences with more than 75 % agreement*
- *Sentences with more than 66 % agreement*
- *Sentences with more than 50 % agreement*

Table S2 gives a detailed overview of the four different agreement specifications. With a falling agreement rate amongst annotators, the relative proportion of sentences declared as positive is slightly increasing. Whereas negative and neutral labelled sentences stay roughly constant. For every agreement rate, the majority of sentences is classified as neutral (about 60%), followed by positive (about26%) and negative (about 14%).
With a decrease in agreement rate, the relative proportions of the three classes are facing only minor changes. However, the number of sentences increases remarkably, as fewer annotators have to agree for a sentence to be included. The slight increase in the positive share can be explained by the difficulties when distinguishing between neutral and positive corporate news.

	% Negative	% Neutral	% Positive	Number of sentences
Sentences with 100%	13.4	61.4	25.2	2259
Sentences with $>$ 75 %	12.2	62.1	25.7	3448
Sentences with $>$ 66 %	12.2	60.1	27.7	4211
Sentences with $>$ 50 %	12.5	59.4	28.2	4840

Table S2: Distribution of labels in phrase bank for four subsets formed based on the strength of majority agreement Malo et al. (2014).

When training a model, the performance largely depends on the training data. Two factors, in particular, can influence the results - the quality of the training data, which increases with a higher agreement rate in this specific dataset: Vice versa, the quantity of data increases with a decline in agreement rate. Statistical models tend to gather more information and learn faster on a substantial dataset.

To determine the best-suited training data for this specific task, the different performances of an identically specified model that was trained on three different agreement rates, can be found in Appendix S11.
The results show a decreasing performance in all relevant metrics for lower agreement rates amongst annotators. An increase in the number of sentences used to train the model does not have a positive influence on the results, as the performance on unseen data is aggravated by a lower agreement rate. Therefore, training data with an agreement rate of 100% is used further on.
Another challenge, when working with the given dataset, is the distribution of the data. Many observations are declared as neutral (about 60%), whereas the minority class of negative sentences only accounts for about 13 %. Training a model on an imbalanced dataset is challenging and might hinder the performance. Hence different methods to overcome this restriction will be introduced (e.g. random over- and undersampling, Synthetic Minority Over-sampling Technique).

3.5.4 Challenges during the training process

Overfitting

A primary goal for any machine learning method is to achieve a high generalization power. High accuracy and precision rates that have been achieved on training data should be reproducible on unseen data. One reason that a model delivers near perfect scores on training data, but cannot transfer those results might lay in the model overfitting to the training data. One can distinguish between two different types of overfitting (Hawkins, 2004):

- Employing a model with more flexibility than it needs. An additional level of complexity that has no influence on the performance or worsens the performance of the model.
- Working with a model that is based on irrelevant components, for example, a polynomial of excessive degree.

Overfitting occurs in the recurrent connections of an RNN. Preventing it has been an essential area of research in NLP. One method to overcome overfitting has already been introduced above. Dropout, dropping individual units from the network, is capable of preventing overfitting. A different, more straightforward approach was introduced by Srivastava et al. (2014).

The training process stops as soon as the model performance on a validation set starts to decrease, or no significant improvements are made. Early stopping can be implemented by a callback function in Python. First, the metric to check for has to be defined (e.g. validation loss). Afterwards, a patience argument has to be passed on to the function. The patience argument specifies the number of epochs without an improvement of the validation loss before training is stopped (Chollet et al., 2015).

Figure 11 shows a graphical method to check for overfitting. Training and validation loss are monitored during each epoch of the training process. An increase in validation loss, while the training loss decreases is a first sign that a model is overfitting. The same holds for a constant validation loss and a decrease in training loss. Optimally, the validation and training loss should both be small with a slightly higher training loss.

(a) LSTM with no dropout.

(b) LSTM with a dropout rate of 40%.

Figure 11: Validation and training loss of differently specified LSTM units (only concerning the dropout rate).

An opposite scenario is possible, as well. In case the validation loss is significantly lower than the training loss, the model is underfitting. As underfitting is not as relevant for the analysis, this section focuses entirely on overfitting.

The LSTM with no dropout (a) starts overfitting after 12 epochs. The validation loss does not show any significant improvement from epoch 12 onwards. On the other hand, the LSTM with a drop out rate of 40% (b) shows the first signs of overfitting at period 72. Though it displays improvement of the validation loss later on until epoch 86, both models perform equally well on the training data. Although, for unseen data, the LSTM with dropout outperforms the model with no dropout. A dropout rate of 40 % is chosen, based on performance metrics.

Imbalanced Classes

One problem that was identified by looking at the dataset from Malo et al. (2014) is the class distribution of the data. The number of neutral observations compared to the number of negative and positive instances is skewed. Figure 12 shows the distribution of the data according to the three classes, neutral, positive and negative. It seems to be easier to find neutral news than positive or negative ones, as roughly 1400 out of the 2189 observations are declared as neutral. Although there are other domains, which are far more imbalanced (e.g. Fraud detection domains with about 2% defrauded credit card accounts per year (TOM FAWCETT, 2016)). Still, without adjusting for the imbalanced nature of the training data, conventional methods lean to be biased in favour of the majority class with low accuracy for the minority class (He and Garcia, 2008). Error rates are optimized, without adjusting for the distribution of the data. This might lead to a trivial classifier that ranks every observation as an instance of the majority class since observations from the minority class are considered to be outliers and are ignored.

Figure 12: Distribution of Malo et al. (2014) training data.

As identifying instances of negative and positive classes is more relevant for this analysis, different methods to account for an imbalanced dataset have to be introduced. Although an accurate way of dealing with it depends highly on the given data, the most common approaches are outlined in the following.

Metrics

When training a model based on an imbalanced dataset, metrics beyond accuracy should be used (Chawla et al., 2002). Such parameters can include recall (relative proportion of positive instances that were classified as such) and precision(relative proportion of positive classifications that are genuinely positive) [4]. Optimizing an algorithm with respect to accuracy might lead to a naive classifier that predicts the majority class every time. For example, a classifier in a two-way classification problem with 10% of instances in class A and 90% of the cases in class B, can reach an accuracy of 90% by merely predicting class A for every observation (Chawla et al., 2002). In some instances, switching the metric during parameter and model selection can be enough to achieve the required performance for detecting the minority class.

Over- and Undersampling

The most popular approaches to account for an imbalanced dataset are different sampling techniques (TOM FAWCETT, 2016):

- Over-sampling the minority class (with replacement):
  Increasing the number of observations of the minority class by randomly duplicating instances from this class. Oversampling leads to more data, which does not mean that the quality of the analysis increases. As already existing cases are copied, the variance of certain variables might decrease, while it is still based on the same type of observations. Furthermore, oversampling duplicates the number of errors (TOM FAWCETT, 2016).

- Under-sampling the majority class:
  Drops a random number of observations from the majority class. Independent variables might seem to have a higher variance, due to the loss of some observations. Depending on the size of the dataset, under-sampling might not be the optimal choice to account for imbalanced data. On a small dataset, the number of observation might not be sufficient anymore to provide significant results.

- SMOTE (Synthetic Minority Oversampling TEchnique): This approach over-samples the minority class by creating synthetic examples in the following way (Chawla et al., 2002): First, the difference between the feature vector of interest and its nearest neighbour is computed. In the next step, the difference is multiplied by a random number (between 0 and 1). Finally, the value is added back to the feature vector of interest

[4] For a more detailed look at the computation of performance metrics, please see 1.

(Hilario, 2010). Figure 13 gives a graphical illustration of the SMOTE technique. One significant advantage of using SMOTE is the creation of "new" observations that are computed randomly and based on existing ones. It results in a more general decision region of the minority class (TOM FAWCETT, 2016).

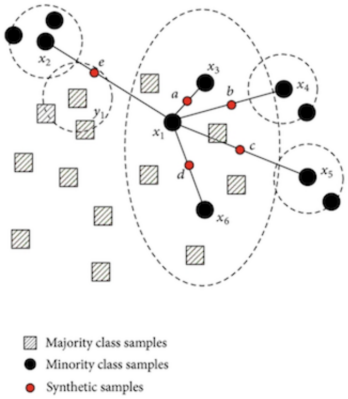


Figure 13: SMOTE oversampling of the minority class (Chawla et al., 2002).

Adjusting Class-weights

Another way to account for an imbalanced dataset is adjusting the class-weights. The machine learning library Scikit-learn (Pedregosa et al., 2011) in python offers the possibility to adjust the "importance" of certain classes (TOM FAWCETT, 2016). The default class_weight parameter of one for each class can be specified. It can be lowered for instances with less importance to the analysis and increased for classes with higher impact on the analysis. The importance of a class is measured by its error. If errors of a class are more costly, the weight can be increased to penalize misclassifications of this instance. For the Malo et al. (2014) dataset, class-weights of positive and negative instances can be increased. Either by manually increasing [5] the class-weights of positives (by a factor of 2.44) and negatives (by a factor of 4.59) or by using the class_weight module of the Scikit-learn library. Both approaches account for the imbalanced character of the data in the same way. As detecting negative and positive instances is more important for this analysis, adjusted class weights will be implemented.

[5]Based on the Malo et al. (2014) dataset with 100% agreement rate. Positives: 570, Negatives: 303 and Neutrals: 1391.

3.5.5 Confusion Matrix

		Accuracy	Precision	Recall	F1-score	Data
LSTM (trainable Wordembeddings)	positive	0.81	0.90	0.76	0.83	672
	neutral	0.84	0.92	0.95	0.94	2961
	negative	0.79	0.84	0.88	0.86	1092
	Weighted avg. / Total	0.80	0.87	0.90	0.90	4725
LSTM (SMOTE Ovesampling)	positive	0.81	0.90	0.84	0.85	2961
	neutral	0.82	0.88	0.90	0.89	2961
	negative	0.81	0.89	0.90	0.89	2961
	Weighted avg. / Total	0.81	0.88	0.88	0.88	8883
LSTM (pre-trained embeddings)	positive	0.79	0.87	0.78	0.82	672
	neutral	0.81	0.86	0.91	0.88	2961
	negative	0.78	0.84	0.87	0.85	1092
	Weighted avg. / Total	0.80	0.86	0.88	0.86	4725
SVM (with Oversampling)	positive		0.78	0.77		2535
	neutral		0.73	0.84		2535
	negative		0.78	0.91		2535
	Weighted avg. / Total		0.76	0.84		7605

Table S3: Performance metrics for differently specified LSTM models and SVM model (see Bommes et al. (2019) for SVM results) based on training data from Malo et al. (2014).

The LSTM models were trained with embedding dimensions of 512, a batch size of 64 and 120 Epochs of training. Hence, an early stopping mechanism was implemented. The model will stop training if the validation loss does not show any improvement for a predefined period, and the best model will be saved. In this case, an early stopping after ten epochs without improvement was chosen. The metrics for optimizing the model were self-defined precision and recall, as the default metric of accuracy tends to overfit the model in situations with imbalanced training data. A dropout rate of 40 % and balanced class weights were implemented to prevent the model from overfitting further. Most of the functions used to define, compile and fit the model were taken from keras (Chollet et al., 2015) and scikit-learn (Pedregosa et al., 2011) [6].

Additionally, the minority classes of positive and negative sentiment were oversampled by using the SMOTE algorithm[7] as part of the imblearn package (Lemaître et al., 2017) for model two. It might seem like the results did not change with SMOTE, but discrimination between neutral-positive and neutral-negative has shown signs of improvement. The overall performance did not benefit from an implementation of pre-trained word embeddings (Word2Vec), and discrimination between groups is worse compared to SMOTE.

Unfortunately, the results from Bommes et al. (2019) did only focus on Precision and Recall of the SVM model. Nevertheless, LSTM shows an improvement over SVM and based on the results. It is better suited for the three-way classification problem at hand. A possible explanation is the ability of LSTM models to account for long term semantic dependencies in text data. Going forward, the final model used for prediction will be the LSTM model with SMOTE oversampling of the minority classes.

[6] Please see the full implementation of the model training under https://github.com/ADoebele/LSTM-Sentiment-Analysis-Coindesk-.

[7] A brief description of SMOTE can be found at 3.5.4.

4 News Data and sentiment construction

This chapter focuses on data, which was gathered from Coindesk (2019) and is used for sentiment prediction. The technique of collecting data with a dynamic Web scraper is emphasised, and necessary steps of preprocessing the data are illustrated. Finally, all relevant information regarding the construction of a sentiment index is given, and the results are briefly analysed.

4.1 Coindesk

Coindesk (2019) is a news website with a particular focus on Blockchain, Bitcoin and Cryptocurrencies as a whole. The site launched in April 2013 and released close to 25000 articles in three different categories (Business, Markets and Technology) since. For the following analysis, the available data in the Business Category, from 8 Apr 2013 up to 8 April 2019, is used. The aim of this thesis is a sentiment analysis based on articles relevant in the context of Blockchain, categories that focus mainly on the price of individual cryptocurrencies are excluded from this research (e.g. markets). The textual data from the source was collected via a dynamic web scraper. It can be used to gather data from all webpages that work with a "load-more" option.[8]Note that requesting data aggressively from a website might lead to breaking the site or getting red flagged by it. Therefore a time.sleep command has been included to mimic human behaviour. One request for one website every second turns out to be sufficient.

In total, there are 7821 articles in the discussed time frame, illustrated in Figure 14. It shows the number of articles per day over the last six years. One can observe an increase in the number of daily articles in 2014 and 2018. Both years are of significance for blockchains most popular use case Bitcoin, as it was undergoing a corrective phase with prices decreasing as much as 71 % in 2014 and 80 % in 2018. Before its corrective state the end of 2013 and 2017 marked periods of price discovery and all-time highs in USD valuation being broken every other day. An increase in blockchain news articles during the same period could indicate a growing interest in the underlying blockchain technology, fueled by the price surge of specific cryptocurrencies.

Additionally, the trading volume of Bitcoin could be related to the number of articles. The growing interest in the blockchain technology might lead to an increase in research and adoption for US companies. An analysis of whether a change in sentiment towards the

[8] https://github.com/ADoebele/LSTM-Sentiment-Analysis-Coindesk/blob/master/DynamicWebscraper(Coindesk).py

blockchain industry influences the returns on US sectors will be performed later on.

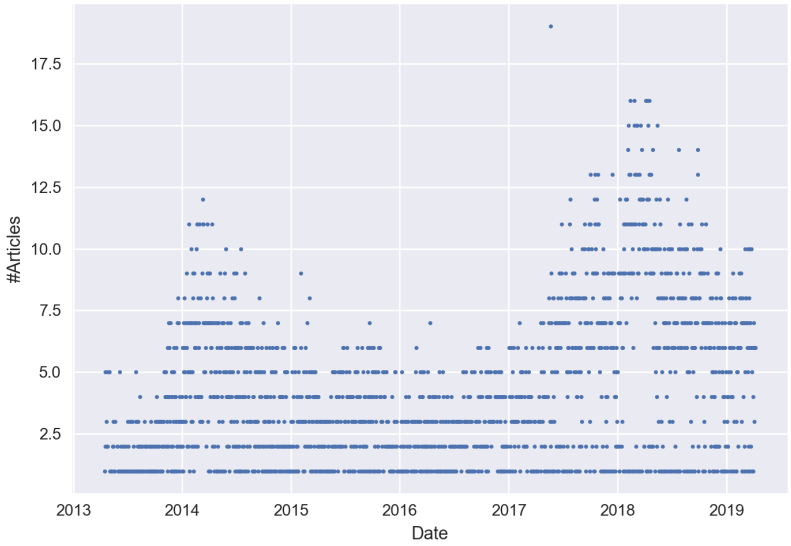


Figure 14: Number of daily articles in the Buisness-section of Coindesk (2019) between March 2013 and Mai 2019.

4.2 Preprocessing of text data

When preprocessing the data, it is essential to assure an identical format for both datasets. The one used to train the model has to have the same form as the one used for prediction. Otherwise, the accuracy of the model might suffer, or the model will generate errors, as vectors can differ in length. As the model was trained on a context-specific dataset, a context coherence of training data and prediction data is assumed. Moreover, processing data should not cause any shrinkage in information content. Assume working with financial data that includes the company Apple. After lowering all words in the data, the company Apple cannot be separated from the fruit. As examples like this only appear rarely, the benefits of lowering words still outweigh the disadvantages.

The training-data from Malo et al. (2014) is already split into sentences; the only thing left to do is lower-case all sentences, keep only those words that are alphabetic and tokenize the resulting words. Afterwards, the data needs to be split into a training and a validation set to prevent overfitting. [9]

[9]Processing training data can be found as part of Model training under https://github.com/ADoebele/LSTM-Sentiment-Analysis-Coindesk-.

Natural language processing tools are used to alter the structure of the data. As the following chapter will carry out a sentence based sentiment prediction, the articles from coindesk.com have to be broken down into a list of sentences for each article. The data returned from the web scraper is a CSV file containing all relevant links from Coindesk.com. Therefore the text has to be extracted from the links gathered in the first step and cleaned afterwards. After converting the links into a list of links, the content of each URL is parsed by using Beautiful Soup. For the construction of a daily sentiment score, the time stamps of each article are collected as well. The Sentiment on days with more than one article has to be calculated by averaging over the sum of sentiment on that given day. After parsing the content, each word is converted to lowercase, and non-alphabetic characters are removed. In the final step, the text is divided into multiple lists of sentences, where each instance corresponds to one article. Both timestamp and content of each article are used to make sentiment based predictions in the following chapter.
Table S4 illustrates the process of cleaning text that has been scraped and parsed from the web[10].

Before processing	After processing
As part of the deal, \xa0Shingo Lavine, founder and CEO of Ethos, will become Voyager's chief innovation officer	as part of the deal shingo lavine founder and ceo of ethos will become voyagers chief innovation officer.

Table S4: Example of processing sentences from Coindesk news.

Besides lowering all characters and removing punctuations, the expression \xa0 is dismissed as well. The dynamic web scraper uses HTLM $< tag >$'s to scrape the Webpage and gather relevant information within the HTML code. Hence remnants of HTML have to be erased. The text cleaning has to be applied to over 50.000 sentences in the dataset, to obtain more accurate predictions from the model.
Lately, the field of NLP has been shifting from bag-of-words models and word encodings towards word embeddings (Brownlee, 2017). We already touched on the benefits of using word

[10]The full implementation of extracting and cleaning text from Coindesk links can be found at https://github.com/ADoebele/LSTM-Sentiment-Analysis-Coindesk-.

embeddings over bag-of-words, as each word is converted into a dense vector that can capture its relative meaning within the text. Also, spelling, punctuations and different variations of the word will automatically be learned in the embedding space. Cleaning the text might therefore not be as significant for the performance of the model as it was with classical NLP (e.g. stemming words). Tomas Mikolov, one of the developers of Word2Vec, said it best when asked how to prepare text data for Word2Vec (Tomas Mikolov, 2015):

> "There is no universal answer. It all depends on what you plan to use the vectors for. In my experience, it is usually good to disconnect (or remove) punctuation from words, and sometimes also convert all characters to lowercase. One can also replace all numbers (possibly greater than some constant) with some single token such as .[...]
>
> All these pre-processing steps aim to reduce the vocabulary size without removing any valuable content (which in some cases may not be true when you lowercase certain words, i.e. 'Bush' is different from 'bush', while 'Another' usually has the same sense as 'another'). The smaller the vocabulary is, the lower is the memory complexity, and the more robustly are the parameters for the words estimated."

Although preprocessing the data in a specific way might not be necessary to train the model, a decrease in vocabulary size carries benefits that can improve the performance of a model. There seems to be no "right" way to process the data, and a lot depends on the task that needs to be performed. When in doubt, one can use differently handled data and compare the metrics for every model result. Based on its scores, the optimal way of altering the input can be found.

4.3 Sentiment index construction

In this step, the "final" LSTM model is used to make predictions on unseen data, which is the Coindesk dataset. Of course, there is no such thing as a perfect model, but for the prediction task, the one that achieved the highest performance metrics is used. As seen in S3 an LSTM model with SMOTE oversampling is the final model for the classification predictions.[11]
As text data is discrete, it is represented in the network by "one hot" encoded input vectors. In general, if there is a total of K text classes, n sentences and class k is provided to the network, the input vector x_i is of length K with zeros throughout except for the k^{th} entry, which is one (Graves, 2013). According to Graves (2013), $Pr(x_i|y_i)$ is multinomial distributed and naturally parameterised by a softmax function [12] at the output-layer in the following way:

$$Pr(x_i = k|y_i) = {y_i}^k = \frac{exp({\hat{y}_i}^k)}{\sum_{k'=1}^{K} exp({\hat{y}_i}^{k'})} \tag{18}$$

The resulting output ${y_i}^k$ is a vector of length $(K \times 1)$. It includes probabilities associated with each class k for a particular sentence i. More precisely, the probabilities that a certain input sentence is part of class k. As the prediction for the coindesk dataset is a three-way classification problem, the derived output vector will be of length (3×1). For each class (positive, neutral and negative), it includes the corresponding probability that a sentence is part of that class. Furthermore, a sentence is predicted to be part of the category with the highest probability. After this step of prediction, the sentences of each article are classified into one of the three groups.

Afterwards, the sentence based sentiment has to be aggregated to conduct the polarity of each article. It is therefore summarized in the following way (Bommes et al., 2019): Each article consists of a series of n sentences with a total of m articles. The vector $S_i = (S_{i1}, S_{i2}, \ldots, S_{in})_{n \in \mathbb{N}}$ contains the sentiment of every sentence for each article i , with $i = 1, \ldots, m$. To obtain article based sentiment, first, the fractions of polarity have to be calculated. For the notation introduced above, this can be done by (Chen et al., 2006):

$$PF_i = n^{-1} \sum_{j=1}^{n} \mathbb{1}(S_{ij} = 1) \quad \text{and} \quad NF_i = n^{-1} \sum_{j=1}^{n} \mathbb{1}(S_{ij} = -1) \tag{19}$$

One can see that PF_i and NF_i denote the positive and negative fraction of sentiment in

[11] Please find the Python implementation of the estimated sentiment index under https://github.com/ADoebele/LSTM-Sentiment-Analysis-Coindesk-.

[12] see softmax activation function at 3.5.2.

one particular article. The larger the number of neutral sentences in an article, the smaller are the corresponding values for PF_i and NF_i. In the next step, an aggregated sentiment measure is introduced. Based on the fractions defined above, it can be notated as (Antweiler and Frank, 2004):

$$B_{iA} = log(1 + PF_i) - log(1 + NF_i) \tag{20}$$

An overall negative polarity is given for $B_{iA} < 0$, which holds if $NF_i > PF_i$. For equal fractions of negative and positive sentiment, the polarity of an article is neutral, $B_{iA} = 0$. Positive polarity can be observed if the fraction of positive sentiment is greater than the fraction of negative sentiment, $B_{iA} > 0$.
Looking at the scale of overall sentiment polarity, one can observe $B_{iA} \in [log(0.5), log(2)]$ because of $PF_i, NF_i \in [0, 1]$. It might be useful to rescale in the following way (Bommes et al., 2019):

$$B_i = log(2)^{-1} B_{iA} \tag{21}$$

For the resulting values, $B_i \in [-1, 1]$ holds. Therefore the interpretation is intuitive. Values closer to -1 indicate negative sentiment in an article, whereas values closer to 1 indicate positive sentiment. Figure 15 shows the daily sentiment estimates for each coindesk article based on the approach above. Note that the sentiment curve was smoothened by aggregating the data on a monthly level.

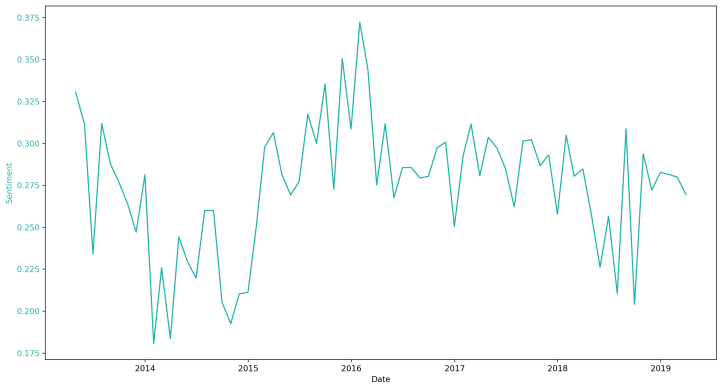


Figure 15: Monthly aggregated sentiment estimates from 2013-04-22 to 2019-03-01.

The average daily sentiment observed over the given period is 0.271507. The overall polarity on a monthly aggregate is positive. This might be either due to an optimistic fundamental outlook on the blockchain technology, or positive biased coverage of blockchain news on Coindesk (2019). Another possibility is a biased estimate based on the poor performance of the model as a result of performing the training process on imbalanced data. Some of these options will be discussed in a bit more detail later on.
The lowest values of sentiment could be observed during 2014 with a further dip in 2018. Both years saw major corrections for blockchains most famous use-case Bitcoin.

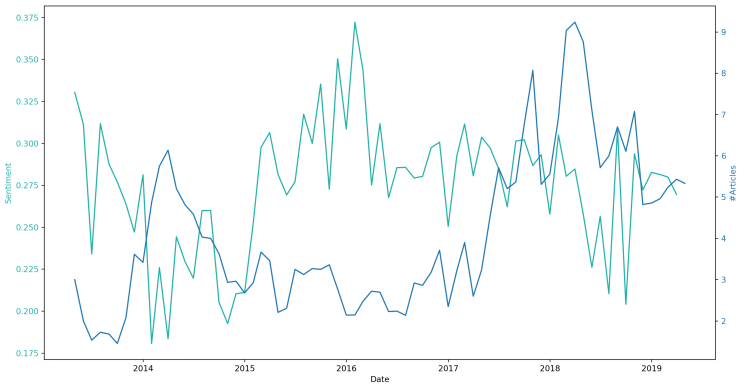


Figure 16: Monthly aggregated sentiment and the number of articles on Coindesk (2019) from 2013-04-22 to 2019-03-01.

Additionally, periods of low sentiment draw much attention to blockchain, as the number of daily articles reaches its highest values during that time. Bearish news seems to be discussed more frequently as high values in sentiment do not lead to an increase in the number of articles that are published (see Figure 16).

5 Applications of the sentiment index

As part of this thesis, the question, how a change in sentiment towards the blockchain technology influences the returns in different US sectors, is answered in this section.
Additionally, the relationship between estimated sentiment in period $t-1$ and the conditional variance of returns in period t is analysed. After generating sentiment estimates for the Coindesk dataset, the indices for eight different US sectors [13] are collected from Spindices (2019). The sector and industry indices measure the performance of the widely-used Global Industry Classification Standard (GICS®) sectors and sub-industries (Spindices, 2019). The financial industry, for instance, consists of firms and institutions that provide financial services to commercial and retail customers (e.g. Banks, investment companies and insurance companies).
According to Ho and Hung (2009), investors sentiment plays a vital role in asset-pricing models. As sentiment often mirrors investors expectations of current market conditions and future developments.

5.1 US Sector Data preprocessing

To analyse the effect of Sentiment changes on the US sector returns, the time horizons for the sector and sentiment data have to match. Therefore some preprocessing of the data is necessary. As Figure 16 has shown, before 2016, the number of daily published articles is quite low. The Sentiment estimates based on the period between 2014 to 2016 might be biased, due to opinions concentrating only on a few instances. In the following analysis, sentiment estimates that refer to the time before 2016 are neglected.[14]. The available data extends from 1 of January 2016 until 01 of March 2019 and contains 795 daily observations of sector indices, sentiment predictions and FARMA5 control variables.
Additionally, the sector indices are transformed into log returns to capture the percentage change for a given day. The benefits of this transformation lay in its normalization character. It is possible to measure variables in a comparable metric and find analytic relationships even though they come from price series with different values. Furthermore, lognormality is given. Under the assumption that prices are distributed lognormally, $log(1+r_t)$ is normally distributed (Merton, 1976).

[13] Those sectors include: Industrial, Communication services, Consumer Discretionary, Consumer Staples, Financials, Health care, Information technology, Energy and real estate.

[14] This task has been carried out in R and can be found at https://github.com/ADoebele/Sector-Sentiment-Preprocessing.

For the given reasons, log returns are chosen in the following analysis.
They are computed in this particular way:

$$r_t = log(\frac{P_t}{P_{t-1}}) = log(P_t) - log(P_{t-1}) \tag{22}$$

with P_t the price of an asset in period t. Accordingly, P_{t-1} for period $t-1$.

5.2 FARMA 5 Control Variables

When analysing dependencies between two variables, the value of the dependent variable might not only dependent on its predecessors in time and a certain independent variable, but it could depend on past values of other variables as well. Therefore control variables need to be introduced to distinguish between direct and indirect effects of sentiment on US sector returns.

The control variables introduced in this thesis are based on research by Fama and French (2015) (Kanuri and McLeod, 2016)[15]:

- SMB_t (Small Minus Big): Return on a diversified portfolio of small stocks minus the return on a diversified portfolio of big stocks for a given time interval t.
- HML_t(High Minus Low): Difference between the returns on diversified portfolios of high and low B/M(Book to market ratio)[16] stocks.
- RMW_t (Robust Minus Weak): Difference between the returns on diversified portfolios of stocks with robust and weak profitability.
- CMA_t (Conservative Minus Aggressive): Difference between the returns on diversified portfolios of the stocks of low and high investment firms.
- $Mkt.RF$ (Excess return on the market): Difference between the value-weight return on all NYSE,AMEX and NASDAQ stocks and the one-month Treasury bill rate.

The controls are used to capture all variation in expected sector returns (Fama and French, 2015) and therefore isolate the effect of an impulse in sentiment on sector returns.

[15]The data can be downloaded under the following link: https://mba.tuck.dartmouth.edu/pages/faculty/ken.french/Data_Library/f-f_5_factors_2x3.html.

[16]A companies book value is calculated by the historical cost of the company. The market value is determined by its share price on the stock market and the number of shares.

5.3 Static contemporaneous Regression

The influence of Sentiment will vary for different US sectors, as not every sector is equally affected by the blockchain technology. One might expect that industries that embrace more blockchain use-cases are better suited for this analysis. The three most promising sectors are finance, information/technology and energy.

The finance sector starts to implement blockchain solutions in numerous ways. Clearing and settlement procedures are primed to be on the blockchain (e.g. SETL, Euroclear and Citi). Another prominent use-case is cross border settlements. Most financial payments need to pass through numerous banks for global transactions, which makes them slow and expensive. The fintech firm Ripple introduced XCurrent, a blockchain based messaging system, to coordinate the transfer of money between banks in a few seconds (Schwartz et al., 2014). Use-cases in the information and technology sector have been growing at a fast rate over the last years, due to rapid blockchain development. The concept of cloud storage is entirely new in its original form, but first attempts of decentralizing cloud storage have been recorded. This includes projects, such as Filecoin (Benet and Greco, 2018), SIA (Ko et al., 2018) and STORJ (Storj Labs, Inc., 2018) . Although the willingness of large tech firms to integrate decentralized cloud storage can be questioned. Another significant aspect is data verification, with promising applications like proof of origin and product quality verification. All those new applications of the blockchain technology might indicate a growing involvement of major US sectors companies in the space.

Furthermore, the Energy sector might benefit from an expansion in blockchain use-cases directly by an increase in energy consumption. The proof-of-work algorithm, which is used for most blockchain applications, is still consuming tremendous amounts of energy. This could benefit the returns in the energy sector.

To get an idea about the relationship between sentiment estimates and the returns in the financial sector, Figure 17 is introduced[17]. It focuses on more recent observations, as the sentiment was estimated based on a more substantial amount of articles per day during that period. The finance returns and sentiment predictions were aggregated on a monthly level to smoothen the series and get a clearer picture of the overall relationship between the two.

They seem to move generally in the same direction, besides the large gap at the beginning around the start of 2016.

[17]The graphs for the IT and energy sector can be found in the Appendix 20, 19.

Figure 17: Estimated monthly aggregated sentiment from 2016-01-01 to 2019-03-01 and returns in the financial sector for the same period.

From the middle of 2016 to the beginning of 2018, both series are moving slightly above their mean. Afterwards, a small decrease in sentiment and returns can be observed until early 2019. With the high fluctuation and no clear trend in the chart, it is hard to establish a relationship between Sentiment and returns in the financial sector based on the graph.
For further analysis of the time series data, a regression model is introduced. A simple static regression model can be written as follows (Wooldridge, 2015):

$$r_t = \beta_0 + \beta_1 S_t + \beta_2 c_t + u_t \tag{23}$$

where the dependent variable r_t denotes the return in the financial sector in period t and S_t the sentiment estimates for the same period. The vector of control variables c_t consists of the variables introduced above. In a static time series regression model, observations of the dependent variable are influenced only by contemporaneous values of the explanatory variables (Wooldridge, 2015). This implies that all interactions between the variables are assumed to occur immediately, or within the same period. Hence, implementing such a model requires the observation interval to be long enough, to allow behavioural adjustments to take place (Wooldridge, 2015). For high-frequency data, this assumption might be violated. For the given reasons, daily observations are aggregated into weekly observations, and a model based on the weekly frequency of observations is implemented. Table S5 contains the regression results based on 165 observations of weekly returns in the financial sector, estimated weekly sentiment and control variables.

Finance	Estimate	Std. Error	t-value	p-value
(Intercept).	0.0008572	0.0007248	1.183	0.239
Sentiment	-0.0029179	0.0025199	-1.158	0.249
Mkt.Rf	0.0110086	0.0003522	31.257	$<$ 2e-16
HML	0.0107513	0.0006139	17.513	$<$ 2e-16
SMB	0.0001074	0.0005588	0.192.	0.848.
RMW	-0.0008734	0.0008325	-1.049	0.296
CMA	-0.0061362	0.0010683	-5.744	4.63e-08
Adjusted R^2	$= 0.9113$			

Table S5: Static regression results for the finance sector.

The results show a high p-value for the coefficient of sentiment, which indicates that it is not significantly different from zero. It can be concluded that the variable sentiment does not have a significant influence on the returns in the financial sector, on an α level of 10% and below. As expected, the control variables Mkt.RF, HML and CMA are highly significant on the 0.1% level. Furthermore, table S6 and S7 show similar results for the Energy and IT sector. The corresponding coefficients are insignificant with high p-values. The regression was carried out in R with the dynlm package (Zeileis, 2019)[18]. Therefore, based on the US sector data and sentiment scores estimated from Coindesk (2019) articles, no dependencies between blockchain sentiment and returns in US sectors could be established.

[18]The R code can be found at https://github.com/ADoebele/Sector-Sentiment-Preprocessing.

IT	Estimate	Std. Error	t-value	p-value
(Intercept).	-6.753e-05	3.809e-03	-0.018	0.985877
Sentiment	4.701e-03	1.277e-02	0.383	0.702037
Mkt.Rf	-2.777e-03	7.889e-04	-3.520	0.000564
HML	-2.061e-03	1.969e-03	-1.046	0.296949
SMB	1.115e-03	1.787e-03	0.624	0.533571
RMW	-2.397e-03	2.351e-03	-1.020	0.309488
CMA	-1.744e-02	2.895e-03	-6.023	1.16e-08
Adjusted R^2	= 0.3292			

Table S6: Static regression results for the Information and Technology sector.

Energy	Estimate	Std. Error	t-value	p-value
(Intercept).	-1.986e-03	4.404e-03	-0.451	0.653
Sentiment	2.303e-02	1.418e-02	1.623	0.107
Mkt.Rf	-2.318e-03	9.123e-04	-2.540	0.012
HML	-9.502e-05	2.277e-03	-0.042	0.967
SMB	6.371e-04	2.067e-03	0.308	0.758
RMW	-1.838e-02	2.719e-03	-6.761	2.51e-10
CMA	-2.336e-03	3.348e-03	-0.698	0.486
Adjusted R^2	= 0.2508			

Table S7: Static regression results for the Energy sector.

5.4 Generalized autoregressive conditional heteroskedasticity (GARCH) model

As seen above, the variable Sentiment does not have any significant influence on the returns of the different sector time series. In this section, the impact of Sentiment on the conditional volatility in returns is analysed. The hypothesis that blockchain sentiment today will increase the conditional volatility in US sector returns tomorrow will be tested by implementing a GARCH model. GARCH models are suitable in cases of heteroskedasticity, hence for data in which error terms are expected to differ in some regions of the data.
Formally heteroskedasticity is defined in the following way (Lütkepohl, 2005) :

$$E[(r_t - \mu)(r_{t-h} - \mu)'] \neq \Gamma_r(h) \neq \Gamma_r(-h)' \quad \text{for all t and h} = 0,1,2,\ldots \tag{24}$$

with returns r_t at time t and r_{t-h} at time t-h. The mean of the time series μ and variance Γ. For different variances at different time intervals, heteroskedasticity is given.
A simple way to check for heteroskedasticity is plotting the time series and look for clusters of volatility.

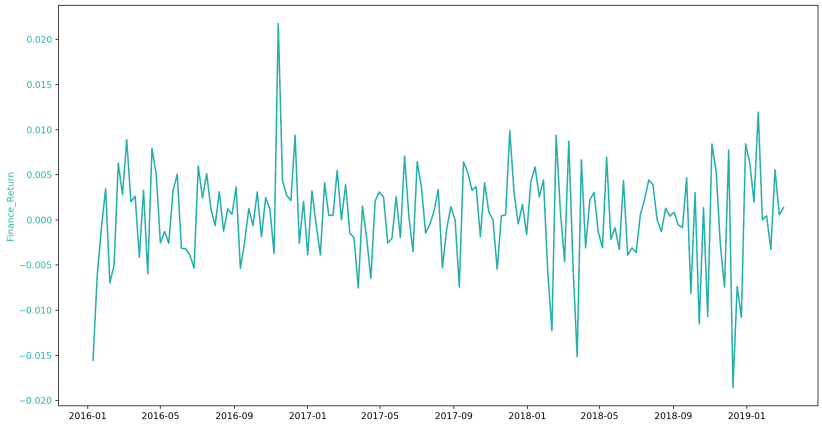


Figure 18: Daily returns in the US finance sector for the period between 2016-01-01 and 2019-03-01.

Figure 18 shows different volatility clusters in the time series of US finance sector returns[19]. The volatility between the beginning of 2016 and 2018 was roughly constant with one significant spike in November 2016. Afterwards, an increase in volatility from the begin-

[19]Please find further plots of US sector time series in the appendix 21,22,23 and 24.

ning of 2018 to 2019 can be observed.

Taking the structure of the time series into account, the implementation of a GARCH model to analyse the influence of Sentiment on volatility in returns is necessary, due to heteroscedasticity. The GARCH model was introduced by Bollerslev (1986) and is a generalized form of the ARCH model that is able to deal with heteroscedasticity. One particular model that has been used in research to depict conditional variance as a function of exogenous variables and its own in financial data is the GARCH(1,1) model (Fratzscher, 2002). Miah and Rahman (2016) have shown that compared to other GARCH(p,q) models, GARCH(1,1) is the best-suited model to explain the volatility of daily stock returns. It expresses the variance at time t in the following form :

$$\sigma_t{}^2 = \omega + \alpha_1 \epsilon_{t-1}{}^2 + \beta_1 \sigma_{t-1}{}^2 \quad \text{with} \quad \epsilon_t = \sigma_t z_t \quad \text{and} \quad z_t \sim N(0,1) \tag{25}$$

where the conditional variance in t depends on the conditional variance in t-1 ($\sigma_{t-1}{}^2$) and the residual returns in t-1 ($\epsilon_{t-1}{}^2$).

To include sentiment as an external regressor, the GARCH(1,1) conditional variance equation has to be updated:

$$\sigma_t{}^2 = \omega + \alpha_1 \epsilon_{t-1}{}^2 + \beta_1 \sigma_{t-1}{}^2 + \gamma_1 S_{t-1} \tag{26}$$

The conditional variance in t now depends on past values of sentiment estimates from t-1. To justify the use of the sentiment term in the conditional variance equation, a GARCH(1,1) model with the sentiment index as an external regressor is estimated in R. For that purpose the R package *rugarch* (Ghalanos, 2019) is used[20].

Table S8 gives an overview of the coefficient estimates based on daily financial sector returns. The coefficient γ is highly significant with an extremely low p-value, manifesting a significant relationship between the conditional variance in financial returns (in period t) and past sentiment values (in period $t-1$) for the blockchain industry although the value of the

[20]The implementation can be found under https://github.com/ADoebele/Coindesk-Sentiment-Application-US-Sectors.

Coefficients	Estimates	std. errors	p-value
	Finance		
ω	0.000010	0.000001	0.000000
α	0.191088	0.048171	0.000073
β	0.715559	0.048013	0.000000
γ	0.00015	0.000001	0.000000

Table S8: Estimated coefficients of GARCH(1,1) model based on daily financial returns.

coefficient is quite small and therefore the strength of the relation relatively weak. A similar picture developed for the sectors Communication, Industrials, Consumer Discretionary and Information and Technology. The results are summarized in Table S9. Extremely low standard errors could explain equal coefficients and high p-values across different sectors. The findings will be further discussed in the conclusions and discussion section.

Coefficients	Estimates	std. errors	p-value
Communication			
ω	0.000010	0.000001	0.000000
α	0.191088	0.048171	0.000073
β	0.715559	0.048013	0.000000
γ	0.00015	0.000001	0.000000
Industrials			
ω	0.000010	0.000001	0.000000
α	0.191088	0.048171	0.000073
β	0.715559	0.048013	0.000000
γ	0.00015	0.000001	0.000000
Consumer discretionary			
ω	0.000010	0.000001	0.000000
α	0.191088	0.048171	0.000073
β	0.715559	0.048013	0.000000
γ	0.00015	0.000001	0.000000
Information and Technology			
ω	0.000010	0.000001	0.000000
α	0.191088	0.048171	0.000073
β	0.715559	0.048013	0.000000
γ	0.00015	0.000001	0.000000

Table S9: Estimated coefficients of GARCH(1,1) model based on daily returns.

6 Conclusions and discussion

This research aimed to produce Sentiment estimates for news articles in the field of Blockchain technology. For that purpose, different LSTM models have been implemented. It could be observed that the overall metrics (Accuracy, F1 score and Precision) did not vary tremendously between the particular models while training on the Malo et al. (2014) dataset. Implementing pre-trained word embeddings (Word2Vec) has not improved the performance significantly compared to the LSTM network with trainable word embeddings. The complexity of the task might not have been high enough to benefit from the additional information gain of pre-trained word embeddings. A sentence based three-way classification problem, like the one at hand does have less potential outcomes than a language modelling task. When predicting the next word in a text document, the number of possible outcomes is way higher, and the use of Word2Vec might be better suited in this case. Additionally, the context of this analysis plays an important role. As Word2Vec was not trained solely on a financial dataset, the model benefits from trainable word embeddings tailormade for business news.

One method that helped to increase the model performance, particularly distinguishing between neutral sentiment and positive - negative sentiment, was oversampling the minority classes. As the negative and positive instances were of greater interest for this research, an increase in performance metrics for those particular cases was desirable. The implementation of SMOTE to balance the number of observations amongst classes, improved the performance of recognizing negative and positive instances, at the equal expense of the neutral class. Even slight changes to the model, like switching the evaluation metrics from accuracy to recall and F1 score had a significant impact on the overall performance.

One key question of this thesis was whether LSTM models are better suited for a sentence based three-way classification problem than SVM. It showed that the LSTM outperformed SVM in the task of sentiment analysis. In the context of NLP, the cell states allow the LSTM to model long term semantic dependencies more efficiently, as past information influences decisions in the next layer. Modelling long term semantic dependencies is especially useful in NLP. Although sentence-level analysis might not make full use of long term semantic dependencies, the results show a significant improvement when using LSTM over SVM. An exciting approach to this topic would be the analysis of a language modelling task, such as speech recognition, and comparing the results of LSTM and SVM. With a longer document

and more potential outcomes, LSTM might benefit further from the efficient use of long term semantic dependencies and results could drift further apart.

The second main question of this thesis was whether sentiment estimates in the blockchain industry influence returns in different US Sectors. The most promising sectors were identified, and a static contemporaneous regression was performed. However, resulting coefficients of sentiment estimates did not show any significance, and the hypothesis had to be rejected on all appropriate α levels. Additionally, a GARCH model was introduced to examine the interaction between sentiment estimates today, and conditional volatility in sector returns tomorrow. Although the coefficient values were rather small and could suffer from including further controls in the model, they showed highly significant p-values. The findings indicate some explanatory power for blockchain sentiment today on conditional volatility in sector returns tomorrow.

Another finding that needs to be addressed is the distribution of the sentiment estimates. It could be observed that most estimates are positive with a mean of 0.27. This could indicate a potential bias towards positive sentiment classification. One possible explanation is a structural bias in the news coverage of Coindesk. Due to close ties to the blockchain community, Coindesk might have an incentive to report positively, indicating an overall positive state of the blockchain market. Another possible explanation lies in the structure of training and prediction data. To achieve similar results during prediction and training, the format of the underlying data has to match. Primarily, similarities in the language used for articles in training and prediction affects results tremendously. This thesis was conducted under the assumption that the financial news from Malo et al. (2014) have similar properties to the Blockchain news articles on Coindesk, and both are defined by a formal language. One could make the case, that the formal language used in financial news differs from the one used in blockchain news, due to different interest groups. This might require additional research to determine whether the structure between the two alters or not. Ultimately, the outlook for blockchain, in general, might be optimistic and therefore, the model generates a majority of positive estimates.

One of the most significant limitations of this work lays in the computational power needed to model deeper neural networks. Every additional layer in a neural network increases the

number of trainable parameters and hence, the flexibility of the model. As for some tasks, a shallow neural network might be sufficient, the addition of further layers in the LSTM model has increased the model performance slightly, while the time consumed for computation increased drastically. In light of the analysis, this result did not justify the addition of various layers at the expense of computational time. At this point, it is hard to tell whether the implementation of multiple LSTM layers would have increased the overall performance significantly. Moreover, conducting Sentiment analysis with deeper LSTM models could be of academic interest.

A further possibility to overcome imbalanced classes that was not covered in this thesis is cost-sensitive learning. In regular learning, all misclassifications are treated equally. With cost-sensitive learning, one can specify the costs of misclassification in a particular class. For tasks where costs associated with misclassifications vary among samples, a higher penalization for misclassification can be customized. Especially for imbalanced datasets, this can increase the "true positive" rate of the minority class. Although, varies methods to account for the imbalanced training data have been implemented, monitoring the effects of cost-sensitive learning on the performance results would be interesting. The question remains, what are the benefits of a model that achieves outstanding performance metrics on known training data, but cannot reproduce those results on unseen data?

Finally, the interaction between blockchain sentiment and volatility in us sector returns demands further research. The GARCH model implemented in this thesis could be expanded by additional variables to isolate the effect of sentiment on volatility in returns (e.g. Sentiment in traditional financial markets). Research from the field of behavioural economics might help to explain the relationship. Kahneman (2011) studied the irrationality of humans and established different "cognitive biases", which may help to explain the influence of sentiment on personal investment choices.

Research, like the one conducted in this thesis, about the influence of blockchain technology on traditional markets, is still in its early stages. To put matters into perspective: The most prominent blockchain use case (bitcoin) has a market cap of around 200 billion dollars[21]. On the other side, the financial sector alone has a market cap of approximately 7

[21] 05.07.2019

trillion dollars. Blockchain and its use-cases are nowhere near mass adoption, and therefore, its influence on traditional markets as of right now is questionable. The impact of Blockchain technology on everyday life tasks is expected to rise over the coming years, and a future analysis might conduct different results as the market cap of further blockchain use cases rises accordingly. Instead of focusing on US sectors, the research carried out in this thesis could be applied to specific firms with stronger ties to the blockchain industry. After identifying those firms, the influence of blockchain sentiment on returns might show a different result, as smaller firms are expected to react faster on the news than US sectors.

In conclusion, the blockchain space is still emerging, and new applications are launched every day. Not all of them will evolve into use-cases adapted by a large number of people, but the relevance of this technology is growing. The results of this thesis concerning returns and conditional volatility in returns on US sectors might suffer from the early stages of blockchain technology. Its development in the future is hard to judge, but with an increase of usability, the relevance of blockchain news for other traditional markets will grow accordingly and therefore might influence the returns in those markets in a much greater way than outlined in this thesis. The same could be said about machine learning applications and neural networks in particular. The last years have seen a tremendous increase in the usage of such algorithms, even for problems that could have been covered easily by less sophisticated methods. Deep neural networks did not reveal their full potential yet, and this research has shown that for NLP tasks with long term semantic dependencies, an LSTM model is well suited to carry out classifications.

References

Aboody, D., S. Levi, and D. Weiss (2018): "Managerial incentives, options, and cost-structure choices," *Review of Accounting Studies*, 1–30.

Alharbi, G. (2016): "Metadiscourse tagging in academic lectures," Ph.D. thesis, University of Sheffield.

Antweiler, W. and M. Z. Frank (2004): "Is all that talk just noise? The information content of internet stock message boards," *The Journal of finance*, 59, 1259–1294.

Aue, A. and M. Gamon (2005): "Customizing sentiment classifiers to new domains: A case study," in *Proceedings of recent advances in natural language processing (RANLP)*, Citeseer, vol. 1, 2–1.

Bayer, J., C. Osendorfer, D. Korhammer, N. Chen, S. Urban, and P. van der Smagt (2013): "On fast dropout and its applicability to recurrent networks," *arXiv preprint arXiv:1311.0701*.

Benet, J. and N. Greco (2018): "Filecoin: A decentralized storage network," *Protoc. Labs*.

Bishop, C. M. et al. (1995): *Neural networks for pattern recognition*, Oxford university press.

Blitzer, J., M. Dredze, and F. Pereira (2007): "Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification," in *Proceedings of the 45th annual meeting of the association of computational linguistics*, 440–447.

Bollerslev, T. (1986): "Generalized autoregressive conditional heteroskedasticity," *Journal of econometrics*, 31, 307–327.

Bommes, E., C. Y.-H. Chen, and W. K. Härdle (2019): "Textual sentiment and sector-specific reaction," *The Journal of finance and data science*.

Britz, D., A. Goldie, M.-T. Luong, and Q. Le (2017): "Massive exploration of neural machine translation architectures," *arXiv preprint arXiv:1703.03906*.

Brownlee, J. (2017): "Deep Learning for Natural Language Processing," *Machine Learning Mystery, Vermont, Australia*, 322.

Chawla, N. V., K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer (2002): "SMOTE: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, 16, 321–357.

Chen, Z., R. T. Daigler, and A. M. Parhizgari (2006): "Persistence of volatility in futures markets," *Journal of Futures Markets: Futures, Options, and Other Derivative Products*, 26, 571–594.

Chesley, P., B. Vincent, L. Xu, and R. K. Srihari (2006): "Using verbs and adjectives to automatically classify blog sentiment," *Training*, 580, 233.

Cho, K., B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio (2014): "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*.

Chollet, F. et al. (2015): "Keras," https://github.com/fchollet/keras.

Chong, N.-Y. and F. Mastrogiovanni (2011): *Handbook of research on Ambient Intelligence and smart environments: trends and perspective*, Information Science Reference.

Chung, J., C. Gulcehre, K. Cho, and Y. Bengio (2014): "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*.

Coindesk (2019): "Blockchain News," https://www.coindesk.com/category/business-news, Last accessed on 2019-7-11.

Collomb, A., C. Costea, D. Joyeux, O. Hasan, and L. Brunie (2014): "A study and comparison of sentiment analysis methods for reputation evaluation," *Rapport de recherche RR-LIRIS-2014-002*.

Dave, K., S. Lawrence, and D. M. Pennock (2003): "Mining the peanut gallery: Opinion extraction and semantic classification of product reviews," in *Proceedings of the 12th international conference on World Wide Web*, ACM, 519–528.

Deng, L., D. Yu, et al. (2014): "Deep learning: methods and applications," *Foundations and Trends® in Signal Processing*, 7, 197–387.

Dunne, R. A. and N. A. Campbell (1997): "On the pairing of the softmax activation and cross-entropy penalty functions and the derivation of the softmax activation function," in *Proc. 8th Aust. Conf. on the Neural Networks, Melbourne*, Citeseer, vol. 181, 185.

Fama, E. F. and K. R. French (2015): "A five-factor asset pricing model," *Journal of financial economics*, 116, 1–22.

Fratzscher, M. (2002): "Financial market integration in Europe: on the effects of EMU on stock markets," *International Journal of Finance & Economics*, 7, 165–193.

Gelbukh, A. (2006): *Computational Linguistics and Intelligent Text Processing: 7th International Conference, CICLing 2006, Mexico City, Mexico, February 19-25, 2006, Proceedings*, vol. 3878, Springer.

Ghalanos, A. (2019): *rugarch: Univariate GARCH models.*, r package version 1.4-1.

Graves, A. (2013): "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*.

Graves, A., A.-r. Mohamed, and G. Hinton (2013): "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing*, IEEE, 6645–6649.

Grossberg, S. (1988): "Nonlinear neural networks: Principles, mechanisms, and architectures," *Neural networks*, 1, 17–61.

Hawkins, D. M. (2004): "The problem of overfitting," *Journal of chemical information and computer sciences*, 44, 1–12.

He, H. and E. A. Garcia (2008): "Learning from imbalanced data," *IEEE Transactions on Knowledge & Data Engineering*, 1263–1284.

Hilario, A. F. (2010): "Sistemas de Clasificacion Basados en Reglas Difusas Linguisticas Aplicadas a Problemas con Clases no Balanceadas," Ph.D. thesis, Universidad de Granada.

Hinton, G. E., S. Osindero, and Y.-W. Teh (2006): "A fast learning algorithm for deep belief nets," *Neural computation*, 18, 1527–1554.

Ho, C. and C.-H. Hung (2009): "Investor sentiment as conditioning information in asset pricing," *Journal of Banking & Finance*, 33, 892–903.

Hochreiter, S. and J. Schmidhuber (1997): "Long short-term memory," *Neural computation*, 9, 1735–1780.

Huang, G., Z. Liu, L. Van Der Maaten, and K. Q. Weinberger (2017): "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 4700–4708.

Janssen, T. M. (2001): "Frege, contextuality and compositionality," *Journal of Logic, Language and Information*, 10, 115–136.

Joachims, T. (2002): *Learning to classify text using support vector machines*, vol. 668, Springer Science & Business Media.

Kahneman, D. (2011): *Thinking, fast and slow*, Macmillan.

Kalchbrenner, N., E. Grefenstette, and P. Blunsom (2014): "A convolutional neural network for modelling sentences," *arXiv preprint arXiv:1404.2188*.

Kanuri, S. and R. W. McLeod (2016): "Sustainable competitive advantage and stock performance: the case for wide moat stocks," *Applied Economics*, 48, 5117–5127.

Kawakami, K. (2008): "Supervised sequence labelling with recurrent neural networks," *Ph. D. thesis*.

Keskar, N. S., D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang (2016): "On large-batch training for deep learning: Generalization gap and sharp minima," *arXiv preprint arXiv:1609.04836*.

Kim, Y. (2014): "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882*.

Kingma, D. P. and J. Ba (2014): "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*.

Ko, A., I. Toman, and Y. Spektor (2018): "Whitepaper (version 0.6)," .

Koppel, M. and J. Schler (2006): "The importance of neutral examples for learning sentiment," *Computational Intelligence*, 22, 100–109.

LeCun, Y., L. Bottou, Y. Bengio, P. Haffner, et al. (1998): "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, 86, 2278–2324.

Lemaître, G., F. Nogueira, and C. K. Aridas (2017): "Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning," *Journal of Machine Learning Research*, 18, 1–5.

Levy, O. and Y. Goldberg (2014): "Dependency-based word embeddings," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, vol. 2, 302–308.

Lin, T., B. G. Horne, P. Tino, and C. L. Giles (1996): "Learning long-term dependencies in NARX recurrent neural networks," *IEEE Transactions on Neural Networks*, 7, 1329–1338.

Liu, B. (2015): *Sentiment analysis: Mining opinions, sentiments, and emotions*, Cambridge University Press.

Liu, B. and L. Zhang (2012): "A survey of opinion mining and sentiment analysis," in *Mining text data*, Springer, 415–463.

Liu, B. et al. (2010): "Sentiment Analysis and Subjectivity." *Handbook of natural language processing*, 2, 627–666.

Liu, P., S. Joty, and H. Meng (2015): "Fine-grained opinion mining with recurrent neural networks and word embeddings," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 1433–1443.

Loughran, T. and B. McDonald (2011): "When is a liability not a liability? Textual analysis, dictionaries, and 10-Ks," *The Journal of Finance*, 66, 35–65.

Luong, M.-T., I. Sutskever, Q. V. Le, O. Vinyals, and W. Zaremba (2014): "Addressing the rare word problem in neural machine translation," *arXiv preprint arXiv:1410.8206*.

Lütkepohl, H. (2005): *New introduction to multiple time series analysis*, Springer Science & Business Media.

Malo, P., A. Sinha, P. Korhonen, J. Wallenius, and P. Takala (2014): "Good debt or bad debt: Detecting semantic orientations in economic texts," *Journal of the Association for Information Science and Technology*, 65, 782–796.

Mao, J., W. Xu, Y. Yang, J. Wang, Z. Huang, and A. Yuille (2014): "Deep captioning with multimodal recurrent neural networks (m-rnn)," *arXiv preprint arXiv:1412.6632*.

Melville, P., W. Gryc, and R. D. Lawrence (2009): "Sentiment analysis of blogs by combining lexical knowledge with text classification," in *Proceedings of the 15th ACM*

SIGKDD international conference on Knowledge discovery and data mining, ACM, 1275–1284.

Merton, R. C. (1976): "Option pricing when underlying stock returns are discontinuous," *Journal of financial economics*, 3, 125–144.

Miah, M. and A. Rahman (2016): "Modelling Volatility of Daily Stock Returns: Is GARCH (1, 1) Enough?" *American Scientific Research Journal for Engineering, Technology, and Sciences (ASRJETS)*, 18, 29–39.

Mikolov, T., M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur (2010): "Recurrent neural network based language model," in *Eleventh annual conference of the international speech communication association.*

Nakamoto, S. et al. (2008): "Bitcoin: A peer-to-peer electronic cash system," .

Nasekin, S. and C. Y.-H. Chen (2018): "Deep learning-based cryptocurrency sentiment construction," *Available at SSRN 3310784*.

Nguyen, D., K. Vo, D. Pham, M. Nguyen, and T. Quan (2012): "A Deep Architecture for Sentiment Analysis of News," .

Pang, B. and L. Lee (2005): "Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales," in *Proceedings of the 43rd annual meeting on association for computational linguistics*, Association for Computational Linguistics, 115–124.

Pang, B., L. Lee, and S. Vaithyanathan (2002): "Thumbs up?: sentiment classification using machine learning techniques," in *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, Association for Computational Linguistics, 79–86.

Pang, B., L. Lee, et al. (2008): "Opinion mining and sentiment analysis," *Foundations and Trends® in Information Retrieval*, 2, 1–135.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay (2011): "Scikit-learn: Machine Learning in Python ," *Journal of Machine Learning Research*, 12, 2825–2830.

Pennington, J., R. Socher, and C. Manning (2014): "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 1532–1543.

Peter Nagy (2018): "LSTM Sentiment Analysis — Keras," https://www.kaggle.com/ngyptr/lstm-sentiment-analysis-keras/notebook, Last accessed on 2019-5-29.

Pontiki, M., D. Galanis, H. Papageorgiou, I. Androutsopoulos, S. Manandhar, A.-S. Mohammad, M. Al-Ayyoub, Y. Zhao, B. Qin, O. De Clercq, et al. (2016): "Semeval-2016 task 5: Aspect based sentiment analysis," in *Proceedings of the 10th international workshop on semantic evaluation (SemEval-2016)*, 19–30.

Riloff, E. and J. Wiebe (2003): "Learning extraction patterns for subjective expressions," in *Proceedings of the 2003 conference on Empirical methods in natural language processing*.

Sadegh, M., R. Ibrahim, and Z. A. Othman (2012): "Opinion mining and sentiment analysis: A survey," *International Journal of Computers & Technology*, 2, 171–178.

Sak, H., A. Senior, and F. Beaufays (2014): "Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition," *arXiv preprint arXiv:1402.1128*.

Salehinejad, H., S. Sankar, J. Barfett, E. Colak, and S. Valaee (2017): "Recent advances in recurrent neural networks," *arXiv preprint arXiv:1801.01078*.

Scheible, C. (2014): "Supervised and semi-supervised statistical models for word-based sentiment analysis," .

Schwartz, D., N. Youngs, A. Britto, et al. (2014): "The ripple protocol consensus algorithm," *Ripple Labs Inc White Paper*, 5, 8.

Shen, Y., X. He, J. Gao, L. Deng, and G. Mesnil (2014): "Learning semantic representations using convolutional neural networks for web search," in *Proceedings of the 23rd International Conference on World Wide Web*, ACM, 373–374.

Socher, R., C. C. Lin, C. Manning, and A. Y. Ng (2011): "Parsing natural scenes and natural language with recursive neural networks," in *Proceedings of the 28th international conference on machine learning (ICML-11)*, 129–136.

Specht, D. F. (1990): "Probabilistic neural networks," *Neural networks*, 3, 109–118.

Spindices (2019): "U.S. Sector Data," https://eu.spindices.com/index-family/us-equity/sector-industry, Last accessed on 2019-6-09.

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014): "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, 15, 1929–1958.

Storj Labs, Inc. (2018): "Storj: A Decentralized Cloud Storage Network Framework," https://github.com/storj/whitepaper, Last accessed on 2019-6-14.

Swan, M. (2015): *Blockchain: Blueprint for a new economy*, " O'Reilly Media, Inc.".

Tang, D., B. Qin, and T. Liu (2016): "Aspect level sentiment classification with deep memory network," *arXiv preprint arXiv:1605.08900*.

Thakkar, H. and D. Patel (2015): "Approaches for sentiment analysis on twitter: A state-of-art study," *arXiv preprint arXiv:1512.01043*.

TOM FAWCETT (2016): "Learning from Imbalanced Classes," https://www.svds.com/learning-imbalanced-classes/, Last accessed on 2019-6-04.

Tomas Mikolov (2015): "Input documents format," https://groups.google.com/forum/#!msg/word2vec-toolkit/jPfyP6FoB94/tGzZxScO0GsJ, Last accessed on 2019-5-25.

Tripathy, A., A. Anand, and S. K. Rath (2017): "Document-level sentiment classification using hybrid machine learning approach," *Knowledge and Information Systems*, 53, 805–831.

Turney, P. D. (2002): "Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews," in *Proceedings of the 40th annual meeting on association for computational linguistics*, Association for Computational Linguistics, 417–424.

Wang, X., W. Jiang, and Z. Luo (2016): "Combination of convolutional and recurrent neural network for sentiment analysis of short texts," in *Proceedings of COLING 2016, the 26th international conference on computational linguistics: Technical papers*, 2428–2437.

Wooldridge, J. M. (2015): *Introductory econometrics: A modern approach*, Nelson Education.

Yang, B., W.-t. Yih, X. He, J. Gao, and L. Deng (2014): "Embedding entities and relations for learning and inference in knowledge bases," *arXiv preprint arXiv:1412.6575*.

Yin, W. (2017): "Deep neural networks for identification of sentential relations," Ph.D. thesis, lmu.

Zaremba, W., I. Sutskever, and O. Vinyals (2014): "Recurrent neural network regularization," *arXiv preprint arXiv:1409.2329*.

Zeileis, A. (2019): *dynlm: Dynamic Linear Regression*, r package version 0.3-6.

Zhang, C., S. Bengio, M. Hardt, B. Recht, and O. Vinyals (2016): "Understanding deep learning requires rethinking generalization," *arXiv preprint arXiv:1611.03530*.

Zhang, Y. and B. Wallace (2015): "A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification," *arXiv preprint arXiv:1510.03820*.

Zhao, L. and A. Zhao (2019): "Sentiment Analysis Based Requirement Evolution Prediction," *Future Internet*, 11, 52.

Zhou, X., X. Wan, and J. Xiao (2016): "Attention-based LSTM network for cross-lingual sentiment classification," in *Proceedings of the 2016 conference on empirical methods in natural language processing*, 247–256.

A Figures

Figure 19: Estimated monthly aggregated sentiment from 2016-01-01 to 2019-03-01 and returns in the energy sector for the same period.

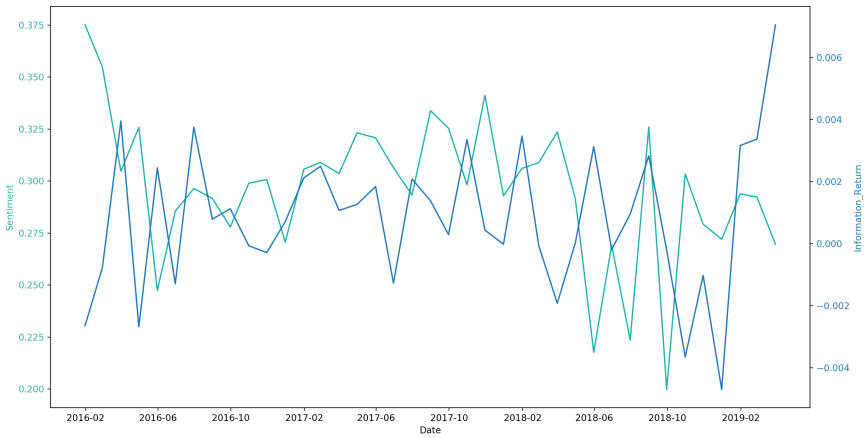


Figure 20: Estimated monthly aggregated sentiment from 2016-01-01 to 2019-03-01 and returns in the Information and Technology sector for the same period.

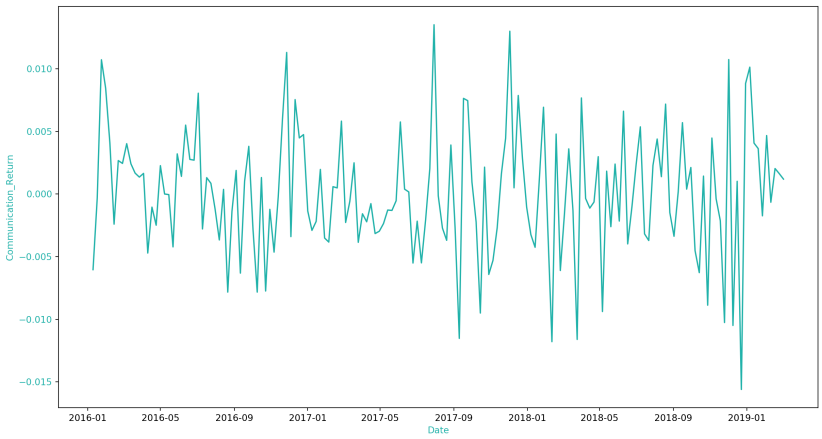


Figure 21: Daily returns in the US Communication sector for the period between 2016-01-01 and 2019-03-01.

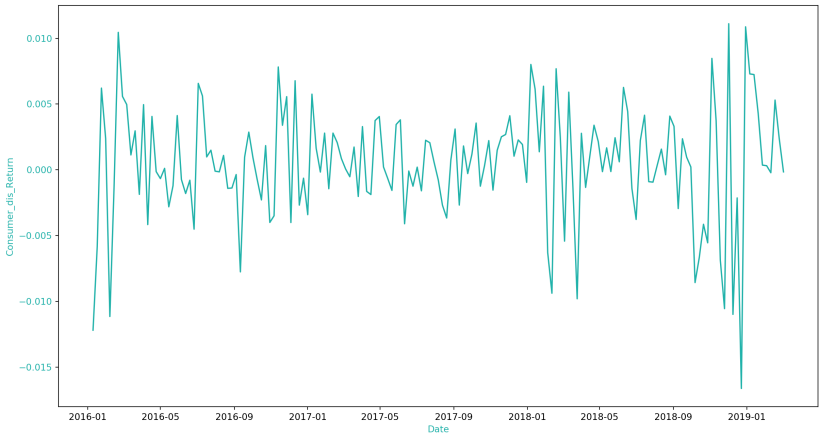


Figure 22: Daily returns in the US Consumer discretionary sector for the period between 2016-01-01 and 2019-03-01.

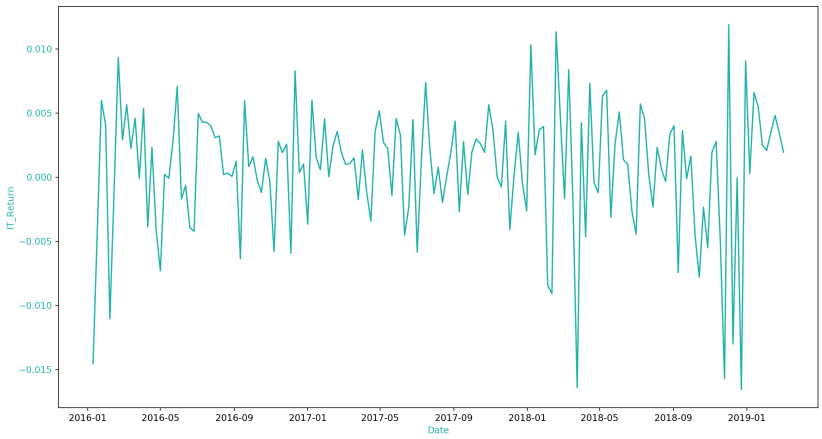


Figure 23: Daily returns in the US Information and Technology sector for the period between 2016-01-01 and 2019-03-01.

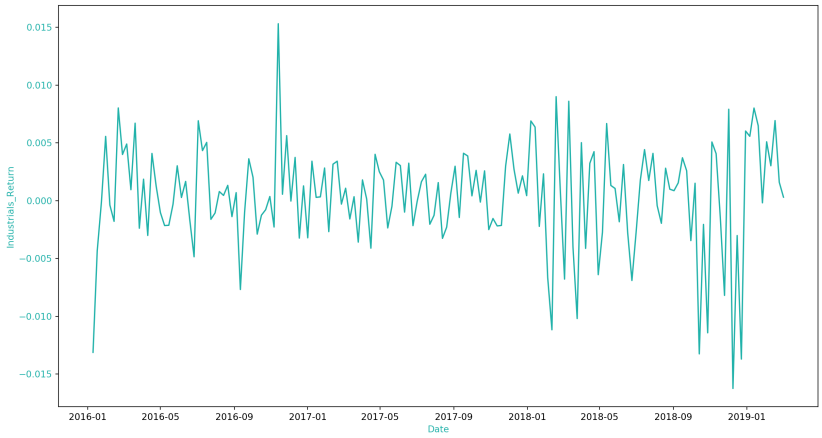


Figure 24: Daily returns in the US Industrial sector for the period between 2016-01-01 and 2019-03-01.

B Tables

	Average pairwise annotator agreement by sentiment category
Number of sentences	150
Number of annotators	16
Overall agreement	0.749
Positive vs. negative	0.987
Negative vs. neutral	0.942
Positive vs. neutral	0.752

Table S10: Interannotator-agreement statistics based on a subset of 150 sentences tagged by all 16 annotators (Malo et al., 2014).

		Accuracy	Precision	Recall	F1-score	Data
LSTM (100% agreement rate)	positive	0.81	0.90	0.76	0.83	672
	neutral	0.84	0.92	0.95	0.94	2961
	negative	0.79	0.84	0.88	0.86	1092
	Weighted avg. / Total	0.80	0.87	0.90	0.90	4725
LSTM (75% agreement rate)	positive	0.77	0.86	0.71	0.78	961
	neutral	0.80	0.88	0.91	0.89	4072
	negative	0.75	0.79	0.84	0.81	1410
	Weighted avg. / Total	0.76	0.82	0.85	0.84	6443
LSTM (50% agreement rate)	positive	0.71	0.79	0.65	0.72	1184
	neutral	0.75	0.82	0.86	0.82	5230
	negative	0.69	0.73	0.78	0.75	1839
	Weighted avg. / Total	0.70	0.76	0.79	0.77	7839

Table S11: Performance metrics for different agreement rates amongst annotators for equalliy specified LSTM networks.

Declaration of Authorship

I hereby confirm that I have authored this Master's thesis independently and without use of others than the indicated sources. All passages which are literally or in general matter taken out of publications or other sources are marked as such.

Berlin, July 12, 2019